WATCHPUG / Pendle v2 / Part 1 Follow up

[H-1] `PendleYearnVaultScy.sol` Wrong decimals for `exchangeRate()`

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/SuperComposableYield/SCY-implementations/PendleYearnVaultScy.sol#L81-L87

```
/**
     * @notice Calculates and updates the exchange rate of shares to underlying asset token
     * @dev It is the price per share of the yvToken
     */
    function exchangeRate() public view override returns (uint256) {
        return IYearnVault(yvToken).pricePerShare();
    }
```

Per the EIP-5115 spec, `exchangeRate()` of SCY is supposed to be:

> exchange rate from SCY token amount into asset amount, scaled by a fixed scaling factor of 1e18.

However, Yearn's `Vault.vy` will take the `decimals` of the underlying token as the `decimals` of the `pricePerShare`:

https://github.com/yearn/yearn-vaults/blob/beff27908bb2ae017ed73b773181b9b93f7435ad/contracts/Vault.vy#L1173

```
@view
@external
def pricePerShare() -> uint256:
    """
    @notice Gives the price for a single Vault share.
    @dev See dev note on `withdraw`.
    @return The value of a single share.
    """
```

https://github.com/yearn/yearn-vaults/blob/beff27908bb2ae017ed73b773181b9b93f7435ad/contracts/Vault.vy#L311-L312

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/libraries/SCY/SCYUtils.sol#L7-L17

```
function scyToAsset(uint256 exchangeRate, uint256 scyAmount) internal pure returns (uint256) {
        return (scyAmount * exchangeRate) / ONE;
    }

    function assetToScy(uint256 exchangeRate, uint256 assetAmount)
        internal
        pure
        returns (uint256)
    {
        return (assetAmount * ONE) / exchangeRate;
    }
```

As a result, when the underlying token's decimals is not `18`, `scyToAsset()` and `assetToScy()` will malfunction and the whole `PendleYearnVaultScy` contract will also be malfunctioning.

For example, `yvUSDC`'s `pricePerShare()` will be about `1e6` instead of the expected `1e18`.

Recommendation

Read and save `IYearnVault.decimals` in the constructor function.

[L-2] `PendleYearnVaultScy.sol` decimals of the SCY token should not be fixed as `18`

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/SuperComposableYield/SCY-implementations/PendleYearnVaultScy.sol#L7-L20

```
contract PendleYearnVaultSCY is SCYBase {
    address public immutable underlying;
    address public immutable yvToken;

    constructor(
        string memory _name,
        string memory _symbol,
        address _yvToken
    ) SCYBase(_name, _symbol, _yvToken) {
        require(_yvToken != address(0), "zero address");
        yvToken = _yvToken;
        underlying = IYearnVault(yvToken).token();
        _safeApprove(underlying, yvToken, type(uint256).max);
    }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/SuperComposableYield/base-implementations/SCYBase.sol#L12-L23

```solidity
abstract contract SCYBase is ISuperComposableYield, PendleERC20, TokenHelper {
    using Math for uint256;

    address public immutable yieldToken;

    constructor(
        string memory _name,
        string memory _symbol,
        address _yieldToken
    ) PendleERC20(_name, _symbol, 18) {
        yieldToken = _yieldToken;
    }
```

> Decimals of the SCY token should reflect the underlying GYGP’s accounting asset’s decimals

The `decimals` of `PendleYearnVaultScy.sol` is now fixed as `18`, while the yvToken's decimals can be different.

[L-2] `PendleQiTokenSCY.sol` decimals of the SCY token should not be fixed as `18`

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/SuperComposableYield/SCY-implementations/BenQi/PendleQiTokenSCY.sol#L12-L28

```solidity
contract PendleQiTokenSCY is SCYBaseWithRewards, PendleQiTokenHelper {
    address public immutable underlying;
    address public immutable QI;
    address public immutable WAVAX;
    address public immutable comptroller;
    address public immutable qiToken;

    constructor(
        string memory _name,
        string memory _symbol,
        address _qiToken,
        address _WAVAX,
        uint256 _initialExchangeRateMantissa
    )
        SCYBaseWithRewards(_name, _symbol, _qiToken)
        PendleQiTokenHelper(_qiToken, _initialExchangeRateMantissa)
    {
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/SuperComposableYield/base-implementations/SCYBaseWithRewards.sol#L11-L21

```
abstract contract SCYBaseWithRewards is SCYBase, RewardManager {
    using Math for uint256;
    using ArrayLib for address[];

    constructor(
        string memory _name,
        string memory _symbol,
        address _yieldToken
    )
        SCYBase(_name, _symbol, _yieldToken) // solhint-disable-next-line no-empty-blocks
    {}
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/SuperComposableYield/base-implementations/SCYBase.sol#L12-L23

```
abstract contract SCYBase is ISuperComposableYield, PendleERC20, TokenHelper {
    using Math for uint256;

    address public immutable yieldToken;

    constructor(
        string memory _name,
        string memory _symbol,
        address _yieldToken
    ) PendleERC20(_name, _symbol, 18) {
        yieldToken = _yieldToken;
    }
```

> Decimals of the SCY token should reflect the underlying GYGP’s accounting asset’s `decimals`

The `decimals` of `PendleQiTokenSCY.sol` is now fixed as `18`, while the QiToken's decimals is `8`.

[H-3] `PendleYearnVaultScy.sol` Wrong implementation of `_redeem()`

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/SuperComposableYield/SCY-implementations/PendleYearnVaultScy.sol#L56-L75

```
function _redeem(address tokenOut, uint256 amountSharesToRedeem)
        internal
        virtual
        override
        returns (uint256 amountTokenOut)
    {

        if (tokenOut == yvToken) {
            amountTokenOut = amountSharesToRedeem;
        } else {
            // tokenOut == underlying
            uint256 sharesRedeemed = IYearnVault(yvToken).withdraw(amountSharesToRedeem);

            require(
                sharesRedeemed != amountSharesToRedeem,
                "Yearn Vault SCY: Not allowed to redeem all shares"
            );

            amountTokenOut = _selfBalance(underlying);
        }
    }
```

1. `IYearnVault.withdraw()` returns the **quantity of tokens redeemed** for `_shares`, not the `sharesRedeemed`.

See: https://github.com/yearn/yearn-vaults/blob/beff27908bb2ae017ed73b773181b9b93f7435ad/contracts/Vault.vy#L1072

2. The `require` condition is wrong: `require(sharesRedeemed != amountSharesToRedeem)` should be `require(sharesRedeemed == amountSharesToRedeem)`.

[M-4] `PendleYieldToken.sol` transfer of YT tokens may revert or cause loss of rewards after expiration

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/core/YieldContracts/PendleYieldToken.sol#L343-L350

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256
) internal override {
    _updateAndDistributeRewardsForTwo(from, to);
    _distributeInterestForTwo(from, to);
}
```

The `_beforeTokenTransfer` hook on `YT` will distribute rewards for both `from` and `to`:

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/libraries/RewardManagerAbstract.sol#L27-L35

```
function _updateAndDistributeRewardsForTwo(address user1, address user2) internal virtual {
    (address[] memory tokens, uint256[] memory indexes) = _updateRewardIndex();
    if (tokens.length == 0) return;

    if (user1 != address(0) && user1 != address(this))
        _distributeRewardsPrivate(user1, tokens, indexes);
    if (user2 != address(0) && user2 != address(this))
        _distributeRewardsPrivate(user2, tokens, indexes);
}
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/core/YieldContracts/PendleYieldToken.sol#L327-L340

```
function _updateRewardIndex()
    internal
    override
    returns (address[] memory tokens, uint256[] memory indexes)
{
    tokens = getRewardTokens();
    if (isExpired()) {
        indexes = new uint256[](tokens.length);
        for (uint256 i = 0; i < tokens.length; i++)
            indexes[i] = postExpiry.firstRewardIndex[tokens[i]];
    } else {
        indexes = ISuperComposableYield(SCY).rewardIndexesCurrent();
    }
}
```

When the `YT` is expired, the reward indexes will be read from `postExpiry`.

However, `PostExpiryData` will only be updated when someone calls a function with the `updateData` modifier, including:

- `mintPY()`
- `redeemPY()`
- `redeemDueInterestAndRewards()`
- `redeemRewardsPostExpiryForTreasury()`

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/core/YieldContracts/PendleYieldToken.sol#L50-L54

```
modifier updateData() {
    if (isExpired()) _setPostExpiryData();
    _;
    _updateScyReserve();
}
```

The a YT holder tries to transfer it after expiration but before anyone calls the functions above, `_updateRewardIndex` will return `indexes` from empty `postExpiry`, which means the index will be `0`.

https://github.com/pendle-finance/pendle-core-internalv2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/libraries/RewardManagerAbstract.sol#L38-L68

```
function _distributeRewardsPrivate(
    address user,
    address[] memory tokens,
    uint256[] memory indexes
) private {
    assert(user != address(0) && user != address(this));

    uint256 userShares = _rewardSharesUser(user);

    for (uint256 i = 0; i < tokens.length; ++i) {
        address token = tokens[i];
        uint256 index = indexes[i];
        uint256 userIndex = userReward[token][user].index;

        if (userIndex == 0) {
            userReward[token][user].index = index.Uint128();
            continue;
        }

        if (userIndex == index) continue;


        uint256 rewardDelta = userShares.mulDown(deltaIndex);
        uint256 rewardAccrued = userReward[token][user].accrued + rewardDelta;

        userReward[token][user] = UserReward({
            index: index.Uint128(),
            accrued: rewardAccrued.Uint128()
        });
    }
}
```

If `userIndex` > 0, then the transaction will revert due to underflow; if the `userIndex` is 0 (the user has never settled the rewards before), then the transcation will go though, and `rewardDelta` is 0, which means the sender will lose the rewards.

Recommendation

Change `_updateRewardIndex()` to:

```
function _updateRewardIndex()
    internal
    override
    returns (address[] memory tokens, uint256[] memory indexes)
{
    tokens = getRewardTokens();
    if (isExpired()) {
        _setPostExpiryData();
        indexes = new uint256[](tokens.length);
        for (uint256 i = 0; i < tokens.length; i++)

            indexes[i] = postExpiry.firstRewardIndex[tokens[i]];
    } else {
        indexes = ISuperComposableYield(SCY).rewardIndexesCurrent();
    }
}
```

A similar issue also applies to `_distributeInterestForTwo()` , and it will be fixed once the fixed above is applied, as `_updateAndDistributeRewardsForTwo()` will be called before `_distributeInterestForTwo()` and `_setPostExpiryData` if needed.

[L-5] `ActionSCYAndYTBase.sol#_swapScyForExactYt()` Wrong return value

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/core/actions/base/ActionSCYAndYTBase.sol#L112-L127

```
function _swapScyForExactYt(
        address receiver,
        address market,
        uint256 exactYtOut,
        uint256 maxScyIn
    ) internal returns (uint256 netScyIn) {
        (, , IPYieldToken YT) = IPMarket(market).readTokens();

        IPMarket(market).swapExactPtForScy(
            address(YT),
            exactYtOut, // exactPtIn = exactYtOut
            _encodeSwapScyForExactYt(msg.sender, receiver, maxScyIn)
        );

        emit SwapYTAndSCY(receiver, exactYtOut.Int(), netScyIn.neg());
    }
```

`netScyIn` is never set within the function, therefore it will always be `0` .

[L-6] `ActionCallback._callbackSwapScyForExactYt()` Wrong implementation to guard against precision issue while calculating `totalScyNeed`

https://github.com/pendle-finance/pendle-core-internal-v2/blob/47470947c7d9c4d9f7bc9efdc549b12059d95cdc/contracts/core/actions/ActionCallback.sol#L79-L113

```
/// @dev refer to _swapScyForExactYt

    function _callbackSwapScyForExactYt(
81          address market,
82          int256 ptToAccount,
83          int256 scyToAccount,
84          bytes calldata data
85      ) internal {
86          VarsSwapScyForExactYt memory vars;
87          (vars.payer, vars.receiver, vars.maxScyToPull) = _decodeSwapScyForExactYt(data);
88          (ISuperComposableYield SCY, , IPYieldToken YT) = IPMarket(market).readTokens();
89
90          ///
91          /// calc totalScyNeed
92          ///
93          SCYIndex scyIndex = SCY.newIndex();
94          uint256 ptOwed = ptToAccount.abs();
95          uint256 totalScyNeed = scyIndex.assetToScy(ptOwed);
96          // to guard against precision issue of lacking a few units of SCY. rawDivUp is not Fix

98
99          ///
100         /// calc netScyToPull
101         ///
102         uint256 scyReceived = scyToAccount.Uint();
103         uint256 netScyToPull = totalScyNeed.subMax0(scyReceived);
104         require(netScyToPull <= vars.maxScyToPull, "exceed SCY in limit");
105
106         ///
107         /// mint & transfer
108         ///
109         SCY.safeTransferFrom(vars.payer, address(YT), netScyToPull);
110
111         uint256 amountPYout = YT.mintPY(market, vars.receiver);
112         require(amountPYout >= ptOwed, "insufficient pt to pay");
113     }
```

At L95, `totalScyNeed` is calculated as:

$$totalScyNeed = \frac{ptOwed \cdot 10^{18}}{exchangeRate_{scy}}$$

The result of the div at L95 should be used directly without any addition when there is no precision loss

For instance:

Given:

- $exchangeRate_{scy}$: 1.1e18
- $ptOwed$: 11e18

Expected results:

- As result of the div at L95 is `10e18`, with no remain (no precision loss), therefore, it should not add `totalScyNeed`
- The final value of `totalScyNeed` should be `10e18`

- The final value of `totalScyNeed` should be 10e18

Actual results:

- L97 will add $exchangeRate_{scy}.divUp(10^{18}) = (1.1 \times 10^{18}).divUp(10^{18}) = 2$ to `totalScyNeed`
- The final value of `totalScyNeed` is 10e18 + 2

When there is a precision loss for the div at L95, the remain 1 wei should be added to `totalScyNeed`

For instance:

Given:

- $exchangeRate_{scy}$: 1.1e18
- $ptOwed$: 10e18

Expected results:

- As the result of the div at L95 is 9090909090909090909, there is a precision loss of 0.0909090909090909... wei, therfore, `totalScyNeed` should add 1 wei to fix the precision loss
- The final value of `totalScyNeed` should be `9090909090909090909 + 1 = 9090909090909090910`

Actual results:

- L97 adds $exchangeRate_{scy}.divUp(10^{18}) = (1.1 \times 10^{18}).divUp(10^{18}) = 2$ to `totalScyNeed`
- The final value of `totalScyNeed` is `9090909090909090909 + 2 = 9090909090909090911`

Recommendation

Consider adding a function named `assetToScyUp()`:

$$scyAmount = (assetAmount \cdot 10^{18}).divUp(exchangeRate_{scy})$$

```
function assetToScyUp(uint256 exchangeRate, uint256 assetAmount)
    internal
    pure
    returns (uint256)
{
    return (assetAmount * ONE).rawDivUp(exchangeRate);
}
```

[H-7] `_swapScyForExactYt()` When current `exchangeRate` is lower than the highest history `exchangeRate`, the user

will be charged for extra SCY and lose part of the PT minted to the market

`scyIndexCurrent()` is the highest history exchangeRate for the SCY:

https://github.com/pendle-finance/pendle-core-internal-v2/blob/47470947c7d9c4d9f7bc9efdc549b12059d95cdc/contracts/core/YieldContracts/PendleYieldToken.sol#L175-L179

```
175 /// @dev maximize the current rate with the previous rate to guarantee non-decreasing rate
176 function scyIndexCurrent() public returns (uint256 currentIndex) {
177     currentIndex = Math.max(ISuperComposableYield(SCY).exchangeRate(), _scyIndexStored);
178     _scyIndexStored = currentIndex.Uint128();
179 }
```

As a result, when the current exchangeRate is lower than the highest history exchangeRate (`scyIndexCurrent()`), at `ActionCallback._callbackSwapScyForExactYt()` L111, `YT.mintPY(market, vars.receiver)` will mint extra some PT to the PtMarket.

`PendleYieldToken.mintPY()` is using `PendleYieldToken.scyIndexCurrent()` to calculate the amounts of PT and YT tokens that can be minted with `scyAmountIn` of SCY.

However, `PendleRouter.swapScyForExactYt()` is using `scy.exchangeRate()` (the current exchange) to calculate the `totalScyNeed` for the desired amount of YT:

$$totalScyNeed = \frac{exactYtOut \cdot 10^{18}}{exchangeRate_{scy}}$$

$exchangeRate_{scy}$ is `SCY.exchangeRate()` , while L111 `YT.mintPY(market, vars.receiver)` is using `PendleYieldToken.scyIndexCurrent()` , highest history exchangeRate of the SCY.

Therefore, when $exchangeRate_{scy}$ < `scyIndexCurrent()` , the `totalScyNeed` calculated will be higher than expected, so that L111 `YT.mintPY(market, vars.receiver)` will mint more PT than expected to the `market` , which constitutes user's loss.

https://github.com/pendle-finance/pendle-core-internal-v2/blob/47470947c7d9c4d9f7bc9efdc549b12059d95cdc/contracts/core/actions/ActionCallback.sol#L79-L113

```
79  /// @dev refer to _swapScyForExactYt
80      function _callbackSwapScyForExactYt(
81          address market,
82          int256 ptToAccount,
83          int256 scyToAccount,
84          bytes calldata data
85      ) internal {
86          VarsSwapScyForExactYt memory vars;
87          (vars.payer, vars.receiver, vars.maxScyToPull) = _decodeSwapScyForExactYt(data);
88          (ISuperComposableYield SCY, , IPYieldToken YT) = IPMarket(market).readTokens();
```

```

        ///
        /// calc totalScyNeed
        ///
        SCYIndex scyIndex = SCY.newIndex();
        uint256 ptOwed = ptToAccount.abs();

        // to guard against precision issue of lacking a few units of SCY. rawDivUp is not Fix
        totalScyNeed += SCYIndex.unwrap(scyIndex).rawDivUp(SCYUtils.ONE);

        ///
        /// calc netScyToPull
        ///
        uint256 scyReceived = scyToAccount.Uint();
        uint256 netScyToPull = totalScyNeed.subMax0(scyReceived);
        require(netScyToPull <= vars.maxScyToPull, "exceed SCY in limit");

        ///
        /// mint & transfer
        ///
        SCY.safeTransferFrom(vars.payer, address(YT), netScyToPull);


        require(amountPYout >= ptOwed, "insufficient pt to pay");
    }
```

Recommendation

Consider using `scyIndexCurrent()` instead of `exchangeRate()` to calculate `totalScyNeed` in `_callbackSwapScyForExactYt()` .

Or, send the extra PT back to the `msg.sender` (`vars.receiver`).

[G-8] Avoid unnecessary expensive external calls can save gas

https://github.com/pendle-finance/pendle-core-internal-v2/blob/27da8ab658a0ba2482a1695a5da1f992f6dafd3d/contracts/core/YieldContracts/PendleYieldToken.sol#L275-L293

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

    if (isExpired()) {
        // post-expiry, all incoming rewards will go to the treasury
        // hence, we can save users one _redeemExternal here
        for (uint256 i = 0; i < tokens.length; i++)
```

```
            postExpiry.userRewardOwed[tokens[i]] -= userReward[tokens[i]][user].accrued;
        rewardAmounts = __doTransferOutRewardsLocal(tokens, user, receiver);

    } else {
        _redeemExternalReward();
        rewardAmounts = __doTransferOutRewardsLocal(tokens, user, receiver);
    }
}
```

In the current implementation, `_redeemExternalReward()` will be called whenever `redeemDueInterestAndRewards()` is called.

However, `ISuperComposableYield(SCY).claimRewards()` is a quite expensive call. If someone has already called `redeemDueInterestAndRewards` recently (since once they do, they actually harvested the rewards for the whole SCY), there is a good chance that the contract itself already has sufficient balance for the claim.

Furthermore, it's unnecessary to transfer fees to the treasury every time.

Consider adding a new function called `__doTransferOutRewardsWithRedeemExternalReward()`:

```
function __doTransferOutRewardsWithRedeemExternalReward(
    address[] memory tokens,
    address user,
    address receiver
) internal returns (uint256[] memory rewardAmounts) {
    uint256 feeRate = IPYieldContractFactory(factory).rewardFeeRate();

    rewardAmounts = new uint256[](tokens.length);

    bool externalRewardRedeemed;
    for (uint256 i = 0; i < tokens.length; i++) {
        uint256 rewardPreFee = userReward[tokens[i]][user].accrued;

        userReward[tokens[i]][user].accrued = 0;

        uint256 feeAmount = rewardPreFee.mulDown(feeRate);
        rewardAmounts[i] = rewardPreFee - feeAmount;

        // NEWLY ADDED STORAGE: treasuryFee
        // instead of push funds to the treasury for every call
        // the admin will need to pull funds; this saves gas
        treasuryFee[tokens[i]] += feeAmount;

        //  redeemExternalReward only if no enough funds
        if (!externalRewardRedeemed && IERC20(tokens[i]).balanceOf(address(this)) < rewardAmou
            _redeemExternalReward();
            externalRewardRedeemed = true;
        }
        _transferOut(tokens[i], receiver, rewardAmounts[i]);
    }
}
```