Pendle V2 - Main contracts

Audit Report

Prepared by Christoph Michel

July 22, 2022.

Contents

1 Introduction

Pendle Finance is a yield tokenisation and yield trading protocol. The documentation for auditors can be found at https://pendle.notion.site.

1.1 Scope of Work

The auditors were provided with a GitHub repository at commit hash 27da8ab. The scope for the audit is the main V2 contracts (excluding the liquidity mining contracts).

The task was to audit the contracts, consisting of the following files with their `sha1` hashes:

File	SHA1
PendleAaveV3SCY.sol	`e661141e9411f3ed1331fd92052016776f4c28a6`
WadRayMath.sol	`98977a834a39a69076a104fb7c19afad421543f8`
PendleQiTokenHelper.sol	`dbc4975389493c1097e2eccfcfe04d49ef10b08b`
PendleQiTokenSCY.sol	`137cb4a6d77e8f5869e665a1e3185126bd579e36`
PendleERC4626SCY.sol	`bf6a16ae8b80d5ccd6a0c0a541bdebc94ae6e988`
PendleWstEthSCY.sol	`58acb4652695c9b17a077c311b507f6a3c94e0f1`
PendleYearnVaultScy.sol	`cfe93d1d07a4145b4a68e8952faee8fe3a401fd9`
SCYBase.sol	`7c1551932bea92abb85db75ba3529cc8e84d3a8e`
SCYBaseWithRewards.sol	`551dbdbd0e9aa1eeb811524830d75cc9b81be2dd`
PendleMarket.sol	`1a35c5c75db5f2e692726319eae15032eef95bcf`
PendleMarketFactory.sol	`eef8cd04737fef0d04abb7e38d15b172f968c547`
PendleERC20.sol	`7025f16fd163339dcf8f215655127e1c03661318`
PendleERC20Permit.sol	`f7a2876f3d799ca36121030a2c7138f3523dbd6f`
PendleRouter.sol	`c23d9022d4a0781cef784f2145a228cf4cc40d5d`

File	SHA1
InterestManagerYT.sol	19deb5aba241977dc6fcd0b3b4548002c956bbe2
PendlePrincipalToken.sol	bd10733632b6625dce6c53e9539113bd83aac793
PendleYieldContractFactory.sol	cf53ff61e8baee7f528b055a55766a6bd0a19c10
PendleYieldToken.sol	a98df96d2151cb69841e37cf37cb43967589b22c
ActionCallback.sol	a090f77eef155db9df58f08e0157353738e13d25
ActionCore.sol	2078591261ddaa605a1e466057f86b9255101902
ActionYT.sol	5c6304b7a3fe69ecb17914e4ba924ce0d6d69136
ActionSCYAndPTBase.sol	2cb1418fdb2fa2717d312d019f6b926a0a5534a6
ActionSCYAndPYBase.sol	23906f8132c38dbee7fc379304d33238750936ce
ActionSCYAndYTBase.sol	851b5c3d16237c10b1d153c9da7b71e7323091d9
CallbackHelper.sol	8650b2705cba4db564c2901058053e9b54a37471
IAToken.sol	f58d583b92d1a8064e085e51f43e1c4e5e01fac7
IAavePool.sol	7fd71319a46b8ccc16e69afb0622f4935caf891f
IAaveRewardsController.sol	f13770fab74c37beceed37ce39916dc2d782242d
IBenQiComptroller.sol	311ab8dd32c7e999e725b7fbdf78c2f5d41d68b2
IBenQiInterestRateModel.sol	c0f295819588e7c48d000f0462a58ce7658db73d
ICelerMessageBus.sol	75793092715916c0b4340d22e4958d6ba859de77
ICelerMessageReceiverApp.sol	547c13c4993cb0b2aa24be94271a141d233b1924
IERC4626.sol	1d48c82b8efd5aa7571e1fd42bdca6486a756a34
IJoePair.sol	8418a65953a03edccbcd709be343d635af5ba5e3
IJoeRouter01.sol	6d90b89b5e09957877ae4076b5ea52d6ab182b6b
IJoeRouter02.sol	a5309b6931d43a143195eb1cea6e45307b31b4b9
IPActionCore.sol	cd2c39cf87829b3d0f7a2a113315d362a5957109
IPActionYT.sol	2c0a345bbf1e68c719d834b3b4f07307d1010b07
IPAllAction.sol	4ebf9f1ba761a18bcb9f8cbb91cab20de5c11edf
IPGaugeController.sol	0a47be80a524a79f5bd00eb4df66b911fd12f013
IPGaugeControllerMainchain.sol	8f926cfbf30794fbe4306edc708da8fdc444897b

File	SHA1
IPGovernanceManager.sol	469e89d860042f36e97133724529cd7c0bdbea51
IPInterestManagerYT.sol	f2f9daf0a309f94defd9216a547003b23ca4c9da
IPMarket.sol	c65712228ba757c073258d95ddeafe1c48dec0dc
IPMarketFactory.sol	b064e6ac845b71470b4e648c2529feb660d6f5e3
IPMarketSwapCallback.sol	30bc4cb79f258037062159b185a8c5c3707ebcd2
IPPermissionsV2Upg.sol	caabed973eca3a99fd0c6b832aea4bc49017e717
IPPrincipalToken.sol	3c2dd78f01c7541f9e9692450cd2a6ea064c86c3
IPRouterStatic.sol	a9a1619b35b62549ff650e15cab858eeefb6a093
IPVeToken.sol	ccbcedc98c5f5b09be5caf3283277c2a03690091
IPVotingController.sol	ad1d30e9e5ccb964ec4cbfad4b08c33110aff226
IPVotingEscrow.sol	70af591c44168b343c80b260e43a630f66b3d58d
IPYieldContractFactory.sol	37be34c3b94e912b21f59393de3f8ed65e995ee1
IPYieldToken.sol	d2d298206535f3292b52e1c6aa3573f4cdab686c
IQiAvax.sol	b74e66bcf3558e8e9dea161ecd52975be25a1a89
IQiErc20.sol	743e700d2eb44887936f548ef1a68b3b87943e51
IQiToken.sol	555ca8f1f93d1bd0edf8bcdd70dbee27a1d32d89
IREDACTEDStaking.sol	86627841787a88691cc98a1bbfc7364ee5491591
IRewardManager.sol	32768c4d7e42e7ac511587bce4963a66ddbe3eed
ISuperComposableYield.sol	1bb3c18a9556232e2090c31370154c781a9c4c94
IWETH.sol	453c46417b1562d40efd5ec0b42f57f7ac53c922
IWXBTRFLY.sol	9ec0dbefc68704e42519be3f4ebe36605d61139d
IWstETH.sol	1a9605a95ef0284d1b108a10c4ef33c7afd86f18
IYearnVault.sol	6e970b4a31f7897009bbdc8f6a203828a53f219c
RewardManager.sol	f2dc9751ff434d2e4b9b000d287dc56985581a30
RewardManagerAbstract.sol	d4073ef9b9e0ff4b55b88ff30be60109d54adfb3
SCYIndex.sol	40996a711e3239ac006cedde40c424e6d97f641f
SCYUtils.sol	7370b15904f1d539789908e579bff53b08c8b5ea

File	SHA1
ArrayLib.sol	0776bb54d08f4d48cfc6eb0396c5dfa1e7318a68
MiniHelpers.sol	7342f3f008baf04496f9ac2234a1cad243d83e08
SSTORE2Deployer.sol	b892f69b464cf508ba411999fb3da5e3f3edafda
TokenHelper.sol	49083149fb1de7698d69496e377475315447c33a
LogExpMath.sol	f07a547f2dfc5e4d20cee86b55442af887c7a628
MarketApproxLib.sol	79a8f3160c0b24f8956b8a7020017cbf9984eb59
MarketMathCore.sol	d354dc01bf1287cd2967707d4f61ba1c748f5833
Math.sol	fb96a2fcc544dc7dfa966b0faba93d08916b538e
WeekMath.sol	581a10206d469379a1dc88af978b7e561439fbbc
PendleJoeSwapHelperUpg.sol	d99d32317e94a2369c1c44321a462ccd96d929c2
PendleGovernanceManager.sol	8017a6cc62c115216b5998d54dc8471f8d82badf
PermissionsV2Upg.sol	969ca63452caa379710c856e5f1186d8b14a6de8

The rest of the repository (including `libraries/solmate/*`, `libraries/helpers/ExpiryUtilsLib.sol`, `libraries/traderjoe/JoeLibrary.sol`, `offchain-helpers/*`) was out of the scope of the audit.

1.2 Security Assessment Methodology

The smart contract's code is scanned both manually and automatically for known vulnerabilities and logic errors that can lead to potential security threats. The conformity of requirements (e.g., specifications, documentation, White Paper) is reviewed as well on a consistent basis.

1.3 Auditors

Christoph Michel

2 Severity Levels

We assign a risk score to the severity of a vulnerability or security issue. For this purpose, we use 4 *severity levels* namely:

MINOR

Minor issues are generally subjective in nature or potentially associated with topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

LOW

Low-severity issues are generally objective in nature but do not represent any actual bugs or security problems. These issues should be addressed unless there is a clear reason not to.

MEDIUM

Medium-severity issues are bugs or vulnerabilities. These issues may not be directly exploitable or may require certain conditions in order to be exploited. If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations that make the system exploitable.

HIGH

High-severity issues are directly exploitable bugs or security vulnerabilities. If unaddressed, these issues are likely or guaranteed to cause major problems or, ultimately, a full failure in the operations of the contract.

3 Discovered issues

3.1 SCY exchange rate for yearn vaults has wrong decimals (`high`)

Context: PendleYearnVaultScy.sol#L86

It's important for the `exchangeRate` to be in 18 decimals.

> "exchangeRate * scyBalance / 1e18 must return the asset balance of the account."
> `ISuperComposableYield`

PendleYearnVaultScy directly returns the yearn vault's `pricePerShare`. However, Yearn V2 vaults return the price in `vault` decimals which are the same decimals as the vault's underlying:

- `pricePerShare = 10**vaultTokenDecimals * token.balanceOf(this)/ totalSupply` and therefore `assetAmount = share * pricePerShare / 10**vault.decimals()` and *not* `assetAmount = share * pricePerShare / 1e18`
- The vault decimals are the same as the underlying token decimals

All `SCYIndex` helpers use `SCYUtils` which multiplies/divides by `ONE = 1e18`. The conversions between SCY and asset are wrong for the yearn vault.

Recommendation

Scale yearn's price per share by `10 ** (18 - vaultTokenDecimals)` and add a fork test with a vault that does not have 18 decimals.

3.2 PendleQiToken uses wrong approximation for exchange rate (`high`)

Context: PendleQiTokenHelper.sol#L46

The `totalReservesNew = qiToken.reserveFactorMantissa()* interestAccumulated + reservesPrior` does not scale down (divide by `1e18`) the fixed-point term `qiToken.reserveFactorMantissa()* interestAccumulated`. New interest inflates the reserves by `1e18`, which decreases the exchange rate. This makes it easy for an attacker to suddenly increase/decrease the exchange rate and even leads to situations where the exchange rate could decrease again (when `interestAccumulated > 0`) whereas the protocol expects it to be non-decreasing.

Recommendation

Fix the `totalReservesNew` computation:

```
uint256 totalReservesNew = qiToken.reserveFactorMantissa() *
    interestAccumulated / 1e18 + reservesPrior;
```

Add tests for lending markets that have not been updated at the current block. Ideally, add differential fuzz tests between Qi's computation and your implementation.

3.3 Market state's `totalScy` is not reduced by transferred-out SCY reserve fees (`high`)

Context: MarketMathCore.sol#L318, PendleMarket.sol#L170

When swapping in the `PendleMarket` a certain amount of SCY is transferred out as fees to the reserve. The math library does *not* reduce the market's `totalScy` by this number.

```
// _setNewMarketStateTrade
market.totalScy = market.totalScy.subNoNeg(netScyToAccount);

// PendleMarket.swapExactPtForScy
IERC20(SCY).safeTransfer(receiver, netScyOut);
IERC20(SCY).safeTransfer(market.treasury, netScyToReserve);

// PendleMarket.swapScyForExactPt
IERC20(PT).safeTransfer(receiver, exactPtOut);
IERC20(SCY).safeTransfer(market.treasury, netScyToReserve);
```

This breaks the important market invariant `market.totalScy.Uint()<= IERC20(SCY).balanceOf(address(this))` after a `swapExactPtForScy` call. The contract does not have enough SCY tokens to pay out all LPs as the percentage is taken on the wrongly inflated `market.totalScy` and not on the balance itself.

Recommendation

Decrease the `market.totalScy` also by `netScyToReserve` when doing *any* swap.

3.4 `swapScyForExactYt` action's return value `netScyIn` is always zero (medium)

Context: ActionSCYAndYTBase.sol#117

The `ActionYT.swapScyForExactYt` function call will always return a `netScyIn` value of zero as it is not set in the _swapScyForExactYt call. Every call appears as if the trader did not have to spend any SCY which could lead to integration issues.

Recommendation

Compute the `netScyIn` value. This could be done similar to `_swapExactYtForScy` using pre-and-post balances of the `msg.sender`?

3.5 Swap PT/YT for exact SCY might return less SCY than expected (medium)

Context: WeekMath.sol#L9

When swapping PT (or YT) for exact SCY an approximation algorithm is used. The approximation algorithm can end up suggesting a `netPtIn` amount that leads to receiving *fewer* than `minScyOut` tokens. This is because the approximation algorithm does not work on SCY amounts but converts it to asset amounts and **rounds down**:

```
// @audit rounding down
vars.minAssetOut = index.scyToAsset(minScyOut);

// ...
// @audit checking it against the minAssetOut, not original minScyOut
if (vars.netAssetOut >= vars.minAssetOut) {
    // ...
    if (isAnswerAccepted) // @audit rounds down a second time (but not
        as relevant as the first)
        return (vars.netPtInGuess, index.assetToScy(vars.netAssetOut));
}
```

This is problematic because users of `swapXforExactY(minY)` functions always assume they receive exactly (or slightly more) than `minY` if the function does not fail and might follow it up by transferring out the exact amount. The transfer would in this case revert as one received slightly less than the expected amount.

Example: Assume the `index = 1.2e18`. User calls `swapPtForExactScy(minScyOut = 123456789)`:

- `vars.minAssetOut = index.scyToAsset(minScyOut)= minScyOut * index / 1e18 = 148,148,146` (rounded down from `.8`)
- Assume the algorithm finds a `vars.netAssetOut == vars.minAssetOut` which might already not be enough to receive the desired SCY amount due to the rounding error on `minAssetOut`.
- It returns `netScyOut = index.assetToScy(vars.netAssetOut)= netAssetOut * 1e18 / index = 123456788` which rounds down a second time. (This rounding error is not as relevant as it does not influence the swap simulation itself.)
- User wanted to receive `minScyOut = 123456789` but this function returned a `netScyOut < minScyOut` and the real swap might indeed lead to receiving less than `minScyOut` due to the first rounding error.

> A similar issue exists in `approxSwapYtForExactScy`.

Recommendation

Consider rounding *up* the minimum asset amount computation.

3.6 `_callbackSwapScyForExactYt` might fail due to rounding issues (medium)

Context: ActionCallback.sol#L95

The `_callbackSwapScyForExactYt` computes an asset amount from the `SCY` debt that needs to be repaid. However, this amount is rounded down and it could be that `totalScyNeed` does not actually cover the SCY debt when minting and repaying the rounded-down asset equivalent.

```
// @audit this should round up to make sure we can cover it
uint256 totalScyNeed = scyIndex.assetToScy(ptOwed);
```

Example: Imagine `ptOwed = 123456789` and `scyIndex = 1.2e18`. Then `totalScyNeed = ptOwed * 1e18 / scyIndex = 123456789 * 1e18 / 1.2e18 = 102,880,657`. But if this SCY amount is minted to PT/YT, the PT amount will be `totalScyNeed * scyIndex / 1e18 = 123456788` due to rounding issues. The debt cannot be repaid and the swap fails.

Recommendation

Consider rounding *up* the `totalScyNeed` to ensure the minted PT can always cover the debt.

3.7 First market LP can be frontrun and lose funds (`medium`)

Context: MarketMathCore.sol#L162

The first market LP provider can be sandwiched by an attacker and lose part of their funds.

Example:

- Assume the market is not initialized yet (`totalSupply()== 0`) and its underlying SCY's exchange rate is `1.0` for demonstration purposes.
- Victim wants to add initial liquidity of `(2e18, 2e18)` calling `ActionCore.addLiquidity(victim, market, scyDesired = 2e18, ptDesired = 2e18, minLpOut = 1)`
- Attacker frontruns adding `(scyDesired = 1e4, ptDesired = 1e22 + 1)`. Then `index.scyToAsset (scyDesired)= 1e4` LP tokens are initially minted. The `totalSupply()` is set to `1e4`.
- When the vicitm transaction is mined they receive: `netLpByPt = (2e18 * 1e4)/ (1e22 + 1)= 2e22 / (1e22 + 1)= 1`, `netLpByScy = (2e18 * 1e4)/ 1e4 = 2e18`. Thus due to `lpToAccount = netLpByPt = 1` they receive only a single LP token.
- Attack withdraws `1e4` out of the `1e4 + 1` LP tokens and receives roughly half of the victim's PT due to rounding issues.

Recommendation

A correct `minLpOut` parameter would protect against this kind of attack. It's important to set this value even for the first deposit.

Currently, the initial LP value only depends on the `scyDesired` value but disregards the `ptDesired` value which makes it easy to perform this attack. A tiny `scyDesired` value can be paired with arbitrarily large `pTDesired` values. Taking the geometric mean (or similar compositions) of both as the initial LP value would mint more LP tokens (`sqrt(1e4 * 1e22)= 1e13`) and require significantly more capital to perform the attack.

3.8 `_callbackSwapYtForScy` reverts with a generic underflow message if flashswap cannot be paid back (`low`)

Context: ActionCallback.sol#L134

The `ActionCallback._callbackSwapYtForScy` function needs to pay back the `scyOwed` debt to the market. If the redeemed SCY does not cover the debt, it reverts with a generic underflow message because of the `uint256 netScyOut = totalScyRedeemed - scyOwed` computation. (The `YT.redeemPY` function does not revert if less than `amounts[0]` is minted to `market`.)

Recommendation

All other callback functions revert with an "insufficient pt to pay" error message if the flashswap debt cannot be paid back. Consider doing the same for `_callbackSwapYtForScy`.

```
require(totalScyRedeemed >= scyOwed, "insufficient SCY to pay");
```

3.9 `_swapScyForExactYt` trade might incur value loss (`low`)

Context: ActionSCYAndYTBase.sol#L120, ActionCallback.sol#L103

Swapping SCY for exact `exactYtOut` YT works as follows:

1. Use `swapExactPtForScy` to flashswap some yet unknown amount of SCY tokens to the YT contract, taking on `exactYtOut` PT debt that needs to be repaid.
2. In the callback, we know the exact SCY amount `scyToAccount` *the YT contract* received. If minting PT & YT would cover the PT debt, we are done and the market receives all the PT, the user receives all the YT from the mint. In case, the received SCY does not cover the debt (`totalScyNeed > scyReceived`) we pull in the difference (remaining amount to cover the debt) from the user.

Note how the *YT contract* is the flashswap target and in the case where the received SCY amount already covers the debt, all minted PT is sent to the market. This PT amount can indeed be larger than the owed debt, leading to a value loss for the trader as PT is gifted to the market. This leftover PT will likely be picked up by bots and traded to SCY.

Recommendation

The case of already receiving enough SCY to cover the PT debt is an unlikely case (as it would lead to arbitrage and receiving free YT) and might not lead to issues in practice.
However, in general, the algorithm seems flawed as one can end up with leftover PT (that is currently gifted the market). One would need to trade the PT to SCY through the market and repeat the algorithm until no more PT was left over.

3.10 `LogExpMath.pow` is not unchecked (`low`)

Context: LogExpMath.sol#L250

The `pow` function body is not wrapped in an unchecked block but should be as it was migrated from an older solidity version that relies on implicit overflow behavior. Using `unchecked` also makes the function more gas efficient.

Recommendation

Wrap the `pow` function body in an `unchecked` block.

3.11 Weeks start on Thursday UTC (`low`)

Context: WeekMath.sol#L9

The `getWeekStartTimestamp` returns the timestamp of the current week's start. However, as unix timestamp 0 was on a Thursday at 0:00 UTC, the weeks in the protocol are also Thursday-aligned which is not intuitive for most people.

Recommendation

Consider offsetting the timestamp (for example, by -3 days) to align the start of the week on a Monday/Sunday.

3.12 Signed integer division rounds up on negative values (`low`)

Context: MarketMathCore.sol#L242

The `PendleMarket` math operates on signed integers and integer divisions on negative numbers essentially round the number up (towards 0) instead of down. (`-2e18 / 1.5e18 ~ -4/3 = -1` instead of rounding down to `-2`.) This might lead to issues if it profits the trader, like when swapping SCY to PT where the negative SCY payment amount is rounded up (the absolute value is rounded down).

```
// @audit netAssetToAccount is negative when account has to pay. (
    swapping SCY to PT)
// @audit the assetToScy computation negates negative numbers and
    rounds down and adds the negative sign back
// @audit which means this rounds towards 0 for negative numbers (
    absolute value decreases)
netScyToAccount = index.assetToScy(netAssetToAccount);
netScyToReserve = index.assetToScy(netAssetToReserve);
```

The trader profits from the rounding error as they need to pay 1 token less.

Recommendation

Consider rounding down correctly on negative numbers. See here for pseudo code.

3.13 `RewardManager` issues with reward tokens that are also yield tokens (`low`)

Context: RewardManager.sol#L44

The reward manager considers all tokens in the contract since the last reward balance as rewards:

```
for (uint256 i = 0; i < tokens.length; ++i) {
    address token = tokens[i];

    // the entire token balance of the contract must be the rewards of
        the contract
    uint256 accrued = _selfBalance(tokens[i]) - rewardState[token].
        lastBalance;
    uint256 index = rewardState[token].index;

    if (index == 0) index = INITIAL_REWARD_INDEX;
    if (totalShares != 0) index += accrued.divDown(totalShares);

    rewardState[token].index = index.Uint128();
    rewardState[token].lastBalance += accrued.Uint128();
}
```

This leads to issues if the reward tokens are also the yield token or base tokens:

- If the yield token is also a reward token, any deposits will count as rewards. **Example:** Imagine the yield token is Compound's cDAI, and a user deposits the base token DAI. It is converted to cDAI. This balance increase will count as a cDAI reward. So does directly depositing cDAI. (Further issues exist if the yield token is a rebasing token where the interest is reflected in an increased balance, like Aave's aDAI.)
- All base tokens that are yield tokens need to immediately be converted to the yield token in `_deposit`. They must not leave a balance increase after depositing, otherwise, they would be counted as rewards. (This is correctly done for the existing SCYs.)

Recommendation

Be aware of these limitations when adding reward tokens.

3.14 Protocol does not support fee-on-transfer tokens (`informational`)

Context: SCYBase.sol#L45

Some ERC20 tokens make modifications to their ERC20's `transfer` or `balanceOf` functions. One type of these tokens is deflationary tokens that charge a certain fee for every `transfer()` or `transferFrom()`. When calling `SCY.deposit` the pre-fee amount is used to calculate the shares for the base token instead of the actually received amount. This could lead to receiving the wrong number of shares, compared to depositing with other base tokens.

Recommendation

Be aware of this limitation when creating SCY tokens with fee-on-transfer tokens as base tokens.

3.15 Reward tokens must not contain duplicates (`informational`)

Context: RewardManager.sol#L66, PendleYieldToken.sol#L287

Duplicate reward tokens are double counted in:

- RewardManager.sol#L66: The `rewardState[token].lastBalance` could be decreased multiple times.
- PendleYieldToken.sol: The `userRewardOwed[token]` would decrease multiple times by the user's accrued balance.

Recommendation

Be aware of these limitations when adding reward tokens. Make sure that reward tokens never contain duplicates.

3.16 Liquidity fragmentation (`informational`)

Context: PendleMarketFactory.sol#L35

Each `(expiry (PT), scalarRoot, initialAnchor)` combination leads to a different market creation. This can lead to many low liquidity markets instead of fewer, more liquid, ones. There can be several markets even for the same token & expiry.

Recommendation

Come up with a strategy to counteract the possibility of liquidity fragmentation. For example, create official markets with sensible parameters for common tokens and endorse them on a frontend.

3.17 Miscellaneous (minor)

- The ISuperComposableYield interface code does not match the specification: According to the specification the `assetInfo` function returns a different tuple `(uint8 assetType, uint8 decimals, bytes info)`. The interface only requires a single `exchangeRate` function instead of `exchangeRateCurrent/exchangeRateStored`. The interface also requires implementations of `yieldToken(), rewardIndexesStored(), rewardIndexesCurrent(), accruedRewards()` functions which are not stated in the specification.
- The ISuperComposableYield interface code does not match the example in the documentation: The documentation has an `assetDecimals` and an `assetId` function. The `yieldToken()` function is also called `underlyingYieldToken` in the example.
- The token pull in `_callbackSwapScyForExactYt` is not always necessary and could be guarded with an `if (netScyToPull > 0)` statement.
- Decoding the callback helper receiver address and not using it is unnecessary and costs more gas. Consider only decoding the `ActionType`.
- Some files import the same files several times, see MarketApproxLib. Consider checking if all imports are required again.
- `MarketApproxLib`'s `logitP`: The `Math.IONE.mulDown(comp.feeRate)` term can be simplified to just `comp.feeRate` as it equals `IONE * comp.feeRate / IONE`.
- The math paper section 4.2.2 points out `n_asset` as a bound for `maxPtIn` but the code takes the minimum of `n_asset` and `n_pt`. An often performed check is `exchangeRate >= 1.0` which means `n_pt >= n_asset`. It's not clear why taking the minimum of `(n_asset, n_pt)` is required.
- The `guessMin` for `approxSwapExactScyForYt` could be set to `maxAssetIn` as `maxScyIn` can always repay at least `maxAssetIn` PT debt. (The algorithms guesses a PT amount and flashswaps it to SCY and then repays it with `maxScyIn` plus the swap-received SCY.) This would help converge faster.
- It's unclear why the function is called `newIndex` - what is *new* about the index? `currentIndex` might be a more appropriate name.
- The `isTimeInThePast` function returns `true` for the current `block.timestamp` - meaning, the now is already in the past. `isNotInFuture` might be a more appropriate name.
- `ArrayLib.padZeroRight` is not used anymore and can be removed. It also does not work for any `length`s over 255 as the loop variable `i` is an `uint8`. It should also only iterate up to `min(inputLength, length)` as the array is already zero-initialized when creating it through Solidity.
- `_transferIn` accepts any amount if the token is `NATIVE`. The function is currently never called in that case but it might be good to either 1) check `msg.value == amount` or revert if `token == NATIVE`.
- `PendleERC20.toUint248`: consider using the more intuitive `require(x <= type(uint248).max)`.
- `PendleYieldContractFactory.initialize` can be frontrun. Make sure to initialize it as soon as possible after contract creation and check if the `initialize` function succeeded. Alternatively, deploy and initialize the contract in a single transaction through a helper factory contract.

- The interest fee and reward fee rates can be changed by governance. It's possible that users are frontrun and a fee that they didn't expect is applied. It's also possible to set a fee to more than 100% which would impact the correct functionality of the protocol. Consider adding a maximum interest and reward fee rate and requiring any rate setters to oblige by these values.
- Some SCY implementations create a second "yield token" storage variable when they could use the base `SCYBase.yieldToken`. `PendleWstEthSCY` uses both `yieldToken` and `wstETH` for `_wstETH`. `PendleYearnVaultSCY` uses both `yieldToken` and `yvToken` for `_yvToken`.
- PendleMarket creation should revert if the PT/YT already expired.
- When adding liquidity directly on the market through `mint` the unused SCY / PT (`desired - used`) is not refunded. The router however pulls in only `scyUsed/ptDesired`.

4 Conclusion

Some integration issues with other protocols have been found that break the correctness of the `SuperComposableYieldToken`s that use these integrations. Some rounding issues have been found throughout the codebase which might interfere with the exactness of router swaps, or favor the user and could, in extreme circumstances, become profitable to abuse.

Pendle's SCY, PT&YT, and market creation is permissionless and the nature of a *permissionless* protocol requires users to not invest in contracts that have been created by malicious tokens or parties. Users need to do their own due diligence on the underlying tokens before using the contracts.

Overall, the documentation and the codebase are of high quality. The AMM specification is very detailed and well written. No judgement can be made on the test suite as it was not available to the auditors. The team is encouraged to add further tests for the raised high-severity issues.

Disclaimer

This audit is based on the scope and snapshot of the code mentioned in the introduction. The contracts used in a production environment may differ drastically. Neither did this audit verify any deployment steps or multi-signature wallet setups. Audits cannot provide a guarantee that all vulnerabilities have been found, nor might all found vulnerabilities be completely mitigated by the project team. An audit is not an endorsement of the project or the team, nor guarantees its security. No third party should rely on the audit in any way, including for the purpose of making any decisions about investing in the project.