Pendle v2 (Part 1, Follow up 2) Audit Report

July 22, 2022

Table of Contents

Summary

This report has been prepared for Pendle v2 (Part 1, Follow up 2) 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**
Codebase	**https://github.com/pendle-finance/pendle-core-internal-v2**
Commit	**2be5ad0da54f76a637ddd6a223d49aa02cb8c07f**
Language	**Solidity**

Audit Summary

Delivery Date	**July 22, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**3**

WP-H1: `PendleYieldToken._beforeTokenTransfer()` should not call `_setPostExpiryData()` without checking if `isExpired()`

High

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/core/YieldContracts/PendleYieldToken.sol#L405

```
400     function _beforeTokenTransfer(
401         address from,
402         address to,
403         uint256
404     ) internal override {
405         _setPostExpiryData();
406         _updateAndDistributeRewardsForTwo(from, to);
407         _distributeInterestForTwo(from, to);
408     }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/core/YieldContracts/PendleYieldToken.sol#L288-L301

```
288     function _setPostExpiryData() internal {
289         PostExpiryData storage local = postExpiry;
290         if (local.firstScyIndex != 0) return; // already set
291 
292         _redeemExternalReward(); // do a final redeem. All the future reward income will belong to the treasury
293 
294         local.firstScyIndex = scyIndexCurrent().Uint128();
295         address[] memory rewardTokens = ISuperComposableYield(SCY).getRewardTokens();
296         uint256[] memory rewardIndexes = ISuperComposableYield(SCY).rewardIndexesCurrent();
297         for (uint256 i = 0; i < rewardTokens.length; i++) {
```

```
298                 local.firstRewardIndex[rewardTokens[i]] = rewardIndexes[i];
299                 local.userRewardOwed[rewardTokens[i]] = _selfBalance(rewardTokens[i]);
300             }
301         }
```

By the first time YT token is transferred, `_setPostExpiryData()` will be called and set `firstScyIndex`, without checking if `isExpired()`.

As a result, `postExpiry.firstScyIndex` will be set much earlier, and the value will be lower than expected.

Recommendation

Change `_updateRewardIndex()` to:

https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/core/YieldContracts/PendleYieldToken.sol#L384-L397

```
384     function _updateRewardIndex()
385         internal
386         override
387         returns (address[] memory tokens, uint256[] memory indexes)
388     {
389         tokens = getRewardTokens();
390         if (isExpired()) {
391             _setPostExpiryData();
392             indexes = new uint256[](tokens.length);
393             for (uint256 i = 0; i < tokens.length; i++)
394                 indexes[i] = postExpiry.firstRewardIndex[tokens[i]];
395         } else {
396             indexes = ISuperComposableYield(SCY).rewardIndexesCurrent();
397         }
398     }
```

Since the storage variable `postExpiry` will be read in `_updateRewardIndex()`, it makes more sense to `_setPostExpiryData()` if `isExpired()` in the context of `_updateRewardIndex()` rather than `_setPostExpiryData()` in the context of `_beforeTokenTransfer()`; the gas overhead will

be minimal as a hot SLOAD of `postExpiry` only costs 100 gas.

It's also more fit semantically when it's done in `_updateRewardIndex()`, the function name sounds like it may update the reward index.

Status

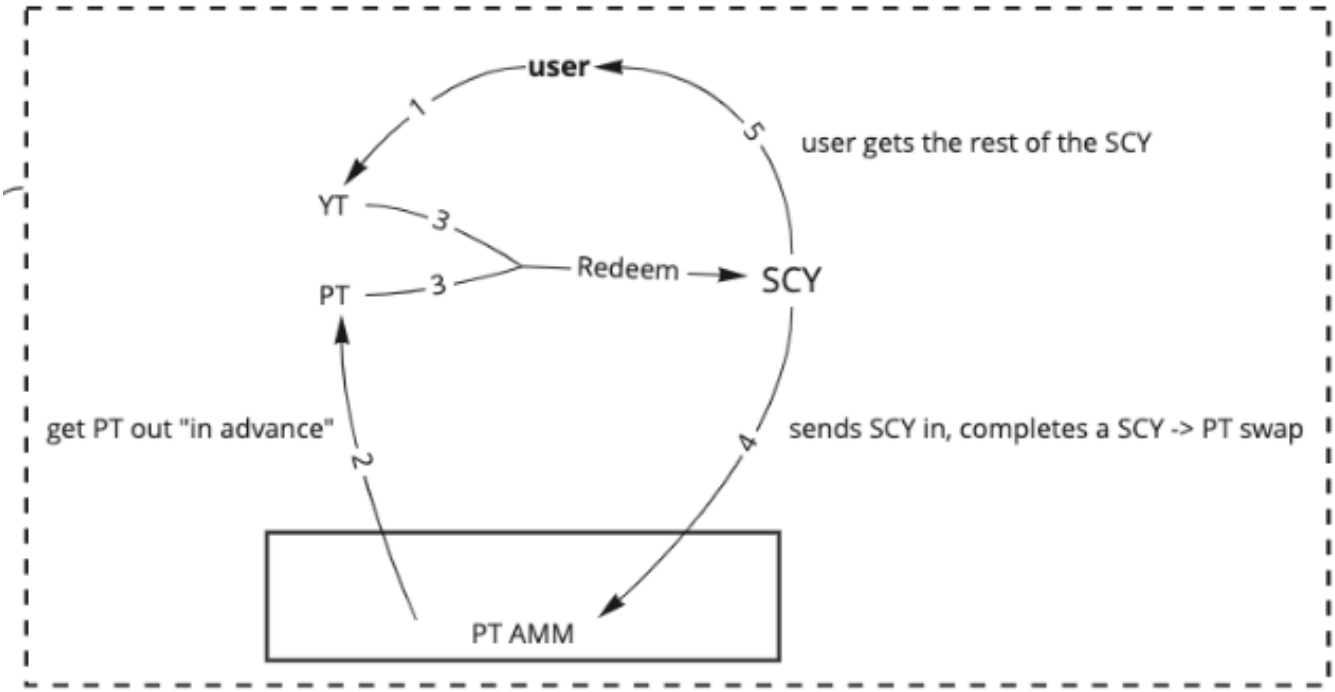
✓ Fixed

WP-M2: `PendleRouter#_callbackSwapYtForScy` may not pay enough SCY to the PT AMM market

Medium

Issue Description

https://pendle.notion.site/How-PendleRouter-works-2c2166bac1784cca81a2c85d796190be



https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/core/actions/ActionCallback.sol#L114-L137

```
114      function _callbackSwapYtForScy(
115          address market,
116          int256 ptToAccount,
117          int256 scyToAccount,
118          bytes calldata data
119      ) internal {
120          (address receiver, uint256 minScyOut) = _decodeSwapYtForScy(data);
121          (ISuperComposableYield SCY, , IPYieldToken YT) =
     IPMarket(market).readTokens();
```

```
122         SCYIndex scyIndex = SCY.newIndex();
123
124         uint256 scyOwed = scyToAccount.neg().Uint();
125
126         address[] memory receivers = new address[](2);
127         uint256[] memory amountPYToRedeems = new uint256[](2);
128
129         (receivers[0], amountPYToRedeems[0]) = (market,
    scyIndex.scyToAssetUp(scyOwed));
130         (receivers[1], amountPYToRedeems[1]) = (
131             receiver,
132             ptToAccount.Uint() - amountPYToRedeems[0]
133         );
134
135         uint256[] memory amountScyOuts = YT.redeemPYMulti(receivers,
    amountPYToRedeems);
136         require(amountScyOuts[1] >= minScyOut, "insufficient SCY out");
137     }
```

Expected:

`market` to repay `scyOwed` SCY;

Actual:

$$\frac{scyAmount \cdot exchangeRate_{scy}}{exchangeRate_{yt.scyIndexCurrent()}}$$

When $exchangeRate_{scy} < exchangeRate_{yt.scyIndexCurrent()}$, the transaction will revert at L216-219 due to ”insufficient SCY” paid to `market` .

https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/libraries/SCY/SCYIndex.sol#L14

```
13     function newIndex(ISuperComposableYield SCY) internal view returns (SCYIndex)
    {
14         return SCYIndex.wrap(SCY.exchangeRate());
15     }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/core/YieldContracts/PendleYieldToken.sol#L224-L248

```
224     function _redeemPY(address[] memory receivers, uint256[] memory
    amountPYToRedeems)
225         internal
226         returns (uint256[] memory amountScyOuts)
227     {
228         uint256 totalAmountPYToRedeem = amountPYToRedeems.sum();
229         IPPrincipalToken(PT).burnByYT(address(this), totalAmountPYToRedeem);
230         if (!isExpired()) _burn(address(this), totalAmountPYToRedeem);
231 
232         uint256 index = scyIndexCurrent();
233         uint256 totalScyInterestPostExpiry;
234         amountScyOuts = new uint256[](receivers.length);
235 
236         for (uint256 i = 0; i < receivers.length; i++) {
237             uint256 scyInterestPostExpiry;
238             (amountScyOuts[i], scyInterestPostExpiry) = _calcScyRedeemableFromPY(
239                 amountPYToRedeems[i],
240                 index
241             );
242             _transferOut(SCY, receivers[i], amountScyOuts[i]);
243             totalScyInterestPostExpiry += scyInterestPostExpiry;
244         }
245         if (totalScyInterestPostExpiry != 0) {
246             postExpiry.totalScyInterestForTreasury +=
    totalScyInterestPostExpiry.Uint128();
247         }
248     }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/core/Market/PendleMarket.sol#L193-L222

```
193     function swapScyForExactPt(
194         address receiver,
195         uint256 exactPtOut,
196         bytes calldata data
197     ) external nonReentrant notExpired returns (uint256 netScyIn, uint256
    netScyToReserve) {
```

```
198         MarketState memory market = readState(true);
199
200         (netScyIn, netScyToReserve) = market.swapScyForExactPt(
201             SCY.newIndex(),
202             exactPtOut,
203             block.timestamp
204         );
205
206         IERC20(PT).safeTransfer(receiver, exactPtOut);
207         IERC20(SCY).safeTransfer(market.treasury, netScyToReserve);
208
209         _writeState(market);
210
211         if (data.length > 0) {
212             IPMarketSwapCallback(msg.sender).swapCallback(exactPtOut.Int(),
    netScyIn.neg(), data);
213         }
214
215         // have received enough SCY
216         require(
217             market.totalScy.Uint() <= IERC20(SCY).balanceOf(address(this)),
218             "insufficient SCY"
219         );
220
221         emit Swap(receiver, exactPtOut.Int(), netScyIn.neg(), netScyToReserve);
222     }
```

Recommendation

Change `ActionCallback._callbackSwapYtForScy()` L122 to: $exchangeRate_{yt.scyIndexCurrent()}$

Status

✓ **Fixed**

WP-I3: Manipulatable `SCY.exchangeRate()` can be dangerous for PY tokens

Informational

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/2be5ad0da54f76a637ddd6a223d49aa02cb8c07f/contracts/core/YieldContracts/PendleYieldToken.sol#L185-L189

```
185  /// @dev maximize the current rate with the previous rate to guarantee
     non-decreasing rate
186  function scyIndexCurrent() public returns (uint256 currentIndex) {
187      currentIndex = Math.max(ISuperComposableYield(SCY).exchangeRate(),
     _scyIndexStored);
188      _scyIndexStored = currentIndex.Uint128();
189  }
```

The design of non-decreasing rate with `scyIndexCurrent()` works pretty well with normal SCYs, even if they have some drawdown, the PY tokens and PT AMM market can still continue to work quite fairly.

However, we find it can be quite dangerous if the SCY's `exchangeRate()` is manipulatable.

For example, if the SCY's exchangeRate is reading the current `balanceOf` underlying token as part of the `totalAsset` or `totalValue` div by the `totalSupply` or `totalShares`, and somehow the `mint()` function allows external calls after the tokens has been transferred in and before the new shares are minted. (Usually enabled by features like swap mint, hookable mint, flash mint, etc.)

In the external call, the attacker can call `PendleYieldToken.sol#scyIndexCurrent()` to update the `_scyIndexStored` to the temporary inflated exchange rate.

And the underlying tokens transferred in to infalte the temporary exchangeRate can be clawback by redeeming the newly minted shares right after.

This would allow the attacker to inflate the `exchangeRate` of the SCY at 0 cost and set the

`scyIndexCurrent()` to an unrealistic high value.

Status

ⓘ Acknowledged

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.