Pendle LP Oracle Audit Report

May 26, 2023

Table of Contents

Summary

This report has been prepared for Pendle LP Oracle 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 LP Oracle**
Codebase	**https://github.com/pendle-finance/pendle-core-v2-public**
Commit	**02a503a849b35482a06003b2c89a7934e81dc02f**
Language	**Solidity**

Audit Summary

Delivery Date	**May 26, 2023**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**4**

[WP-I1] The units of the variables involved in calculating `rateHypTrade` should be consistent.

Informational

Issue Description

> This only impacts the SYs that have fluctuating exchange rates rather than monotonically increasing ones.

https://github.com/pendle-finance/pendle-core-v2-public/blob/02a503a849b35482a06003b2c89a7934e81dc02f/contracts/oracles/PendleLpOracleLib.sol#L77-L96

```
77      function _getPtRates(
78          IPMarket market,
79          MarketState memory state,
80          uint32 duration
81      )
82          private
83          view
84          returns (
85              int256 rateOracle,
86              int256 rateLastTrade,
87              int256 rateHypTrade
88          )
89      {
90          rateOracle = Math.IONE.divDown(market.getPtToAssetRate(duration).Int());
91          rateLastTrade = MarketMathCore._getExchangeRateFromImpliedRate(
92              state.lastLnImpliedRate,
93              state.expiry - block.timestamp
94          );
95          rateHypTrade = (rateLastTrade + rateOracle) / 2;
96      }
```

- The unit for `rateOracle` is how many PT each Asset corresponds to.
- The unit for `rateLastTrade` is how many PT each accounting/virtual asset (corresponding to `py.pyIndexCurrent()`) corresponds to.

Recommendation

`rateLastTrade` and `rateOracle` at L95 should be in the same unit.

Additionally, consider clearly distinguishing between real assets and accounting/virtual assets in naming to improve readability and avoid similar issues.

For example, change L90 to use the new function `market.getPtToAccountingAssetRate()` .

And the `PendlePtOracleLib#getPtToAssetRate()` can be renamed to `PendlePtOracleLib#getPtToRealAssetRate()` .

Status

ⓘ **Acknowledged**

[WP-I2] `getPtToAssetRate()` will revert during the first `duration`

Informational

Issue Description

https://github.com/pendle-finance/pendle-core-v2-public/blob/02a503a849b35482a06003b2c89a7934e81dc02f/contracts/oracles/PendlePtOracleLib.sol#L15-L43

```
function getPtToAssetRate(IPMarket market, uint32 duration)
    internal
    view
    returns (uint256 ptToAssetRate)
{
    uint256 expiry = market.expiry();
    if (expiry <= block.timestamp) {
        return _getPtToAssetRatePostExpiry(market);
    }
    uint256 lnImpliedRate = _getMarketLnImpliedRate(market, duration);
    uint256 timeToExpiry = expiry - block.timestamp;
    uint256 assetToPtRate = uint256(
        MarketMathCore._getExchangeRateFromImpliedRate(lnImpliedRate, timeToExpiry)
    );

    ptToAssetRate = Math.ONE.divDown(assetToPtRate);
}

function _getMarketLnImpliedRate(IPMarket market, uint32 duration)
    private
    view
    returns (uint256)
{
    uint32[] memory durations = new uint32[](2);
    durations[0] = duration;

    uint216[] memory lnImpliedRateCumulative = market.observe(durations);
    return (lnImpliedRateCumulative[1] - lnImpliedRateCumulative[0]) / duration;
}
```

https://github.com/pendle-finance/pendle-core-v2-public/blob/02a503a849b35482a06003b2c89a7934e81dc02f/contracts/core/Market/OracleLib.sol#L118-L149

```
function getSurroundingObservations(
    Observation[65535] storage self,
    uint32 target,
    uint96 lnImpliedRate,
    uint16 index,
    uint16 cardinality
) public view returns (Observation memory beforeOrAt, Observation memory atOrAfter) {
    // optimistically set before to the newest observation
    beforeOrAt = self[index];

    // if the target is chronologically at or after the newest observation, we can early return
    if (beforeOrAt.blockTimestamp <= target) {
        if (beforeOrAt.blockTimestamp == target) {
            // if newest observation equals target, we're in the same block, so we can ignore atOrAfter
            return (beforeOrAt, atOrAfter);
        } else {
            // otherwise, we need to transform
            return (beforeOrAt, transform(beforeOrAt, target, lnImpliedRate));
        }
    }

    // now, set beforeOrAt to the oldest observation
    beforeOrAt = self[(index + 1) % cardinality];
    if (!beforeOrAt.initialized) beforeOrAt = self[0];

    // ensure that the target is chronologically at or after the oldest observation
    if (target < beforeOrAt.blockTimestamp)
        revert Errors.OracleTargetTooOld(target, beforeOrAt.blockTimestamp);

    // if we've reached this point, we have to binary search
    return binarySearch(self, target, index, cardinality);
}
```

It is worth noticing that before the first `duration` has passed, the Oracle will revert with the

error `OracleTargetTooOld` .

Status

ⓘ Acknowledged

[WP-M3] While converting the PTs in the `totalHypotheticalAsset` to Asset, `rateOracle` should be used instead of `rateLastTrade` .

Medium

Issue Description

The goal of the LP Oracle is to prevent manipulation. In order to achieve that, instead of taking the current proportion (p) which is largely impacted by the last trade, it will apply a hypothetical trade that transforms the current state to the expected state based on the Oracle-implied rate. This gives us a trustworthy proportion (p').

However, while the current implementation is using the transformed proportion (p'), it is still using the manipulatable rate (`rateLastTrade`) from the last trade for the conversion from PT to Asset.

As a result, a large part of the `totalHypotheticalAsset` is still open to manipulation.

Another way to think about the issue is:

If we consider that the hypothetical trade actually happened, then the conversion rate from PT to Asset would also NOT be the exchange rate (e) prior to the hypothetical trade, but the exchange rate after that (e').

https://github.com/pendle-finance/pendle-core-v2-public/blob/02a503a849b35482a06003b2c89a7934e81dc02f/contracts/oracles/PendleLpOracleLib.sol#L21-L54

```
function getLpToAssetRate(IPMarket market, uint32 duration)
    internal
    view
    returns (uint256 lpToAssetRate)
{
    MarketState memory state = market.readState(address(0));
    MarketPreCompute memory comp = _getMarketPreCompute(market, state);

    int256 totalHypotheticalAsset;
    if (state.expiry <= block.timestamp) {
        // 1 PT = 1 Asset post-expiry
```

```
        totalHypotheticalAsset = state.totalPt + comp.totalAsset;
    } else {
        (int256 rateOracle, int256 rateLastTrade, int256 rateHypTrade) = _getPtRates(
            market,
            state,
            duration
        );
        int256 cParam = LogExpMath.exp(
            comp.rateScalar.mulDown((rateOracle - comp.rateAnchor))
        );

        int256 tradeSize = (cParam.mulDown(comp.totalAsset) - state.totalPt).divDown(
            Math.IONE + cParam.divDown(rateHypTrade)
        );

        totalHypotheticalAsset =
            comp.totalAsset -
            tradeSize.divDown(rateHypTrade) +
            (state.totalPt + tradeSize).divDown(rateLastTrade);
    }

    lpToAssetRate = _calcLpPrice(totalHypotheticalAsset, state.totalLp).Uint();
}
```

Recommendation

Change to:

```
function getLpToAssetRate(IPMarket market, uint32 duration)
    internal
    view
    returns (uint256 lpToAssetRate)
{
    MarketState memory state = market.readState(address(0));
    MarketPreCompute memory comp = _getMarketPreCompute(market, state);

    int256 totalHypotheticalAsset;
    if (state.expiry <= block.timestamp) {
        // 1 PT = 1 Asset post-expiry
```

```
        totalHypotheticalAsset = state.totalPt + comp.totalAsset;
    } else {
        (int256 rateOracle, int256 rateLastTrade, int256 rateHypTrade) = _getPtRates(
            market,
            state,
            duration
        );
        int256 cParam = LogExpMath.exp(
            comp.rateScalar.mulDown((rateOracle - comp.rateAnchor))
        );

        int256 tradeSize = (cParam.mulDown(comp.totalAsset) - state.totalPt).divDown(
            Math.IONE + cParam.divDown(rateHypTrade)
        );

        totalHypotheticalAsset =
            comp.totalAsset -
            tradeSize.divDown(rateHypTrade) +
            (state.totalPt + tradeSize).divDown(rateOracle);
    }

    lpToAssetRate = _calcLpPrice(totalHypotheticalAsset, state.totalLp).Uint();
}
```

Status

✓ Fixed

[WP-I4] When the real index (`SY.exchangeRate()`) is less than `YT.pyIndexCurrent()` , PendleLpOracleLib may not be working as expected

Informational

Issue Description

> This only impacts the SYs that have fluctuating exchange rates rather than monotonically increasing ones.

https://github.com/pendle-finance/pendle-core-v2/blob/02a503a849b35482a06003b2c89a7934e81dc02f/contracts/oracles/PendleLpOracleLib.sol#L21-L54

```
function getLpToAssetRate(IPMarket market, uint32 duration)
    internal
    view
    returns (uint256 lpToAssetRate)
{
    MarketState memory state = market.readState(address(0));
    MarketPreCompute memory comp = _getMarketPreCompute(market, state);

    int256 totalHypotheticalAsset;
    if (state.expiry <= block.timestamp) {
        // 1 PT = 1 Asset post-expiry
        totalHypotheticalAsset = state.totalPt + comp.totalAsset;
    } else {
        (int256 rateOracle, int256 rateLastTrade, int256 rateHypTrade) = _getPtRates(
            market,
            state,
            duration
        );
        int256 cParam = LogExpMath.exp(
            comp.rateScalar.mulDown((rateOracle - comp.rateAnchor))
        );

```

```
43          int256 tradeSize = (cParam.mulDown(comp.totalAsset) - state.totalPt).divDown(
44              Math.IONE + cParam.divDown(rateHypTrade)
45          );
46
47          totalHypotheticalAsset =
48              comp.totalAsset -
49              tradeSize.divDown(rateHypTrade) +
50              (state.totalPt + tradeSize).divDown(rateLastTrade);
51      }
52
53      lpToAssetRate = _calcLpPrice(totalHypotheticalAsset, state.totalLp).Uint();
54  }
```

`getLpToAssetRate()` is expected to return the value of LP token in terms of Asset.

However, if the historical highest exchange rate of SY is higher than the current actual exchange rate from SY to Asset, then the result will be higher than the expected result.

Recommendation

Consider converting `totalHypotheticalAsset` to actual asset near L53.

Additionally, `PendlePtOracleLib.getPtToAssetRate()` near L30 should also convert units to actual asset.

https://github.com/pendle-finance/pendle-core-v2/blob/02a503a849b35482a06003b2c89a7934e81dc02f/contracts/oracles/PendlePtOracleLib.sol#L15-L31

```
15  function getPtToAssetRate(IPMarket market, uint32 duration)
16      internal
17      view
18      returns (uint256 ptToAssetRate)
19  {
20      uint256 expiry = market.expiry();
21      if (expiry <= block.timestamp) {
22          return _getPtToAssetRatePostExpiry(market);
23      }
24      uint256 lnImpliedRate = _getMarketLnImpliedRate(market, duration);
```

```
    uint256 timeToExpiry = expiry - block.timestamp;
    uint256 assetToPtRate = uint256(
        MarketMathCore._getExchangeRateFromImpliedRate(lnImpliedRate, timeToExpiry)
    );

    ptToAssetRate = Math.ONE.divDown(assetToPtRate);
}
```

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.