Pendle Contract Security Audit

Table of Contents

Scope

The scope of the audit is limited to https://github.com/pendle-finance/contracts/. As the review was conducted over time, with some code and design changes made during this review, the initial commit was 5116f44cb0828d74b77ebfb14f394d61103112bb, and the final commit was 4499f36bd58736b1b0fc48f77ffed19f04ad9308.

Summary of Findings

In performing a security audit of the Pendle, several issues of concern were found. For each finding, a summary of the issue is documented, along with any other finer details regarding the issue. Security recommendations are also provided where applicable.

The table below shows a breakdown of security findings found categorized by severity or risk and impact. A finding that has been reported is listed as pending, and if that finding is satisfactorily mitigated, it will be categorized as fixed.

Severity	**Fixed**	**Pending**	**Total**
Critical	1	0	1
High	0	0	0
Medium	3	0	3
Low	3	0	3
Info	2	0	2

Issues

PEN-001: redeemUnderlying lacks access control check

Severity: Critical
Status: Resolved

Function redeemUnderlying in PendleForgeBase lacks the onlyRouter modifier, thus allowing any address to call the function. A malicious actor could directly call this function, set the _to to his own address and obtain the underlying amount of another user.

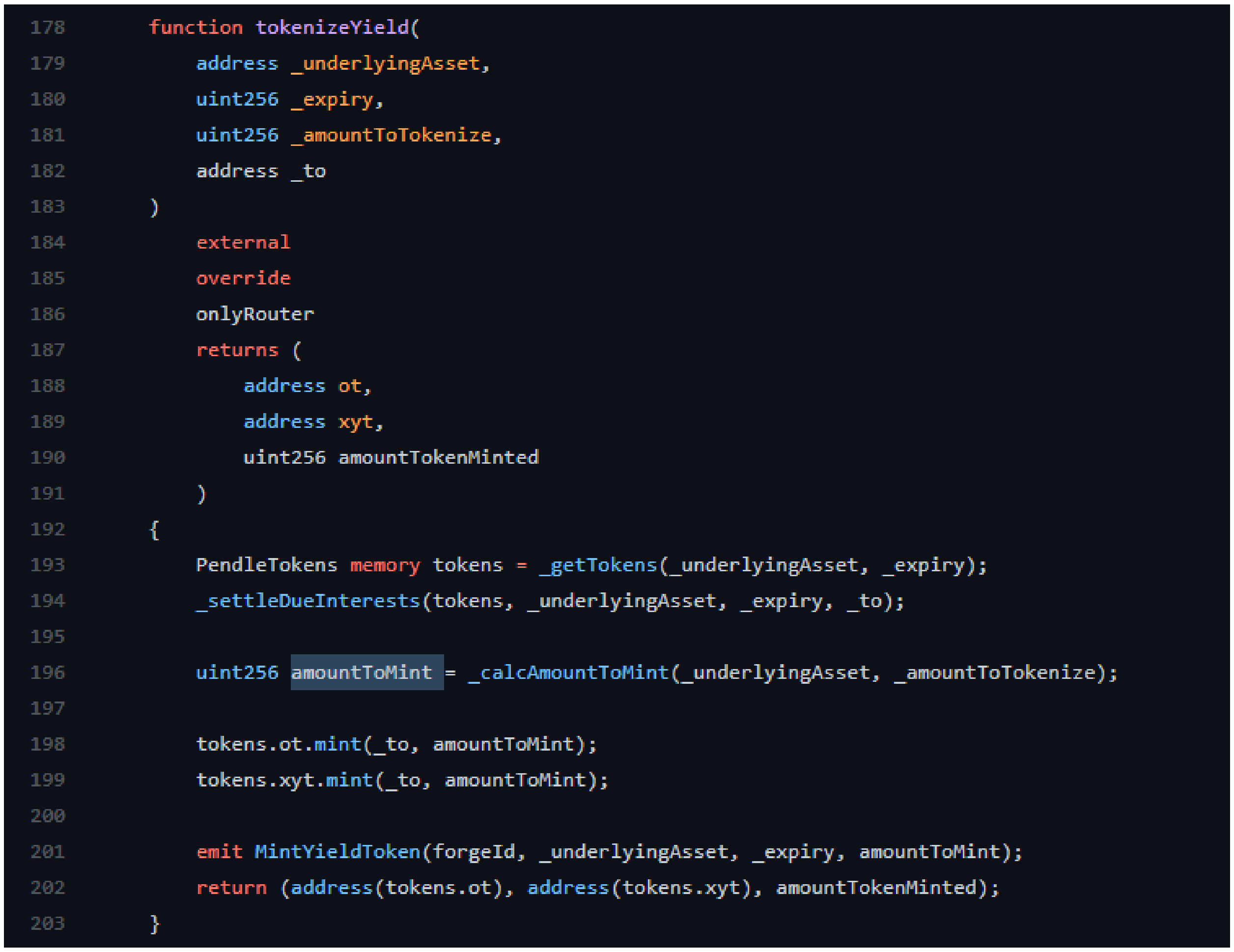
Recommendations
Add the onlyRouter modifier to the function.

This issue has been resolved here.

PEN-002: Improper assigning of return parameter in tokenizeYield

Severity: Medium
Status: Resolved

Function tokenizeYield in PendleForgeBase is supposed to return amountTokenMinted but the function only assigns results to the local variable amountToMint.



```
    function tokenizeYield(
        address _underlyingAsset,
        uint256 _expiry,
        uint256 _amountToTokenize,
        address _to
    )
        external
        override
        onlyRouter
        returns (
            address ot,
            address xyt,
            uint256 amountTokenMinted
        )
    {
        PendleTokens memory tokens = _getTokens(_underlyingAsset, _expiry);
        _settleDueInterests(tokens, _underlyingAsset, _expiry, _to);

        uint256 amountToMint = _calcAmountToMint(_underlyingAsset, _amountToTokenize);

        tokens.ot.mint(_to, amountToMint);
        tokens.xyt.mint(_to, amountToMint);

        emit MintYieldToken(forgeId, _underlyingAsset, _expiry, amountToMint);
        return (address(tokens.ot), address(tokens.xyt), amountTokenMinted);
    }
```

As a result, the returned value will always be 0.

This issue was found by the Pendle team during the course of this audit.

Recommendations
Change the local variable name from amountToMint to amountTokenMinted.

This issue has been resolved here.

PEN-003: Rewards not added up in claimRewards

Severity: Medium
Status: Resolved

Function claimRewards in PendleLiquidityMiningBase didn't add up the rewards the user would receive but instead kept re-assigning the variable.

Recommendations

Add the amount of rewards for each iteration of the loop.

This issue has been resolved here.

PEN-004: Interest compounded after XYT expiry is not withdrawable

Severity: Medium
Status: Resolved

Function _calcDueInterests in PendleAaveForge and PendleAaveV2Forge didn't calculate the compound interest of the user's XYT interest. In the case where the XYT expires and the user does not withdraw the interest immediately after the expiry (let this amount be termed alpha), the amount the user will receive when withdrawing a year later will still be alpha (without the compound interest generated by alpha). Any compound interest generated by alpha will be forever stuck in the forge.

This issue was found by the Pendle team during the course of this audit.

Recommendations
Change the functionality to account for the interest after expiry.

This issue has been resolved here.

PEN-005: Excessive use of gas in claimRewards and claimLpInterests

Severity: Low
Status: Resolved

Functions claimRewards and claimLpInterests in PendleLiquidityMiningBase looped over unbounded data structure, therefore may consume an excessive amount of gas. Also, since the expiries of an user would never get pruned, there would be a lot of redundant claiming in these 2 functions.

This issue was found by the Pendle team during the course of this audit.

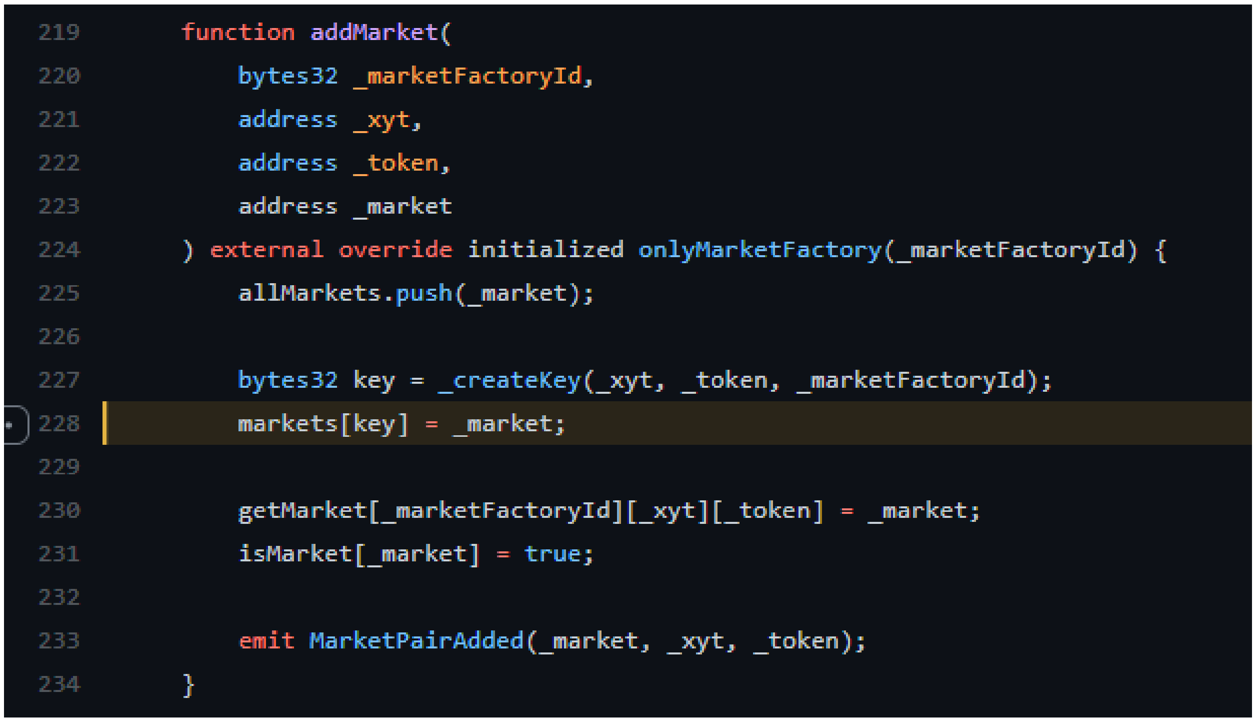
Recommendations
Allow these 2 functions to take in the exact expiry to claim interests & rewards from.

This issue has been resolved here and here.

PEN-006: Lack of duplicate checks of market key after keyCreation

Severity: Low
Status: Resolved

Function addMarket in PendleData generates a mapping key with _createKey when adding a market.



```
    function addMarket(
        bytes32 _marketFactoryId,
        address _xyt,
        address _token,
        address _market
    ) external override initialized onlyMarketFactory(_marketFactoryId) {
        allMarkets.push(_market);

        bytes32 key = _createKey(_xyt, _token, _marketFactoryId);
        markets[key] = _market;

        getMarket[_marketFactoryId][_xyt][_token] = _market;
        isMarket[_market] = true;

        emit MarketPairAdded(_market, _xyt, _token);
    }
```

In the unlikely case of a key collision, an existing market's address would be overwritten.

Recommendations
Add a check that a created key does not already exist before using it.

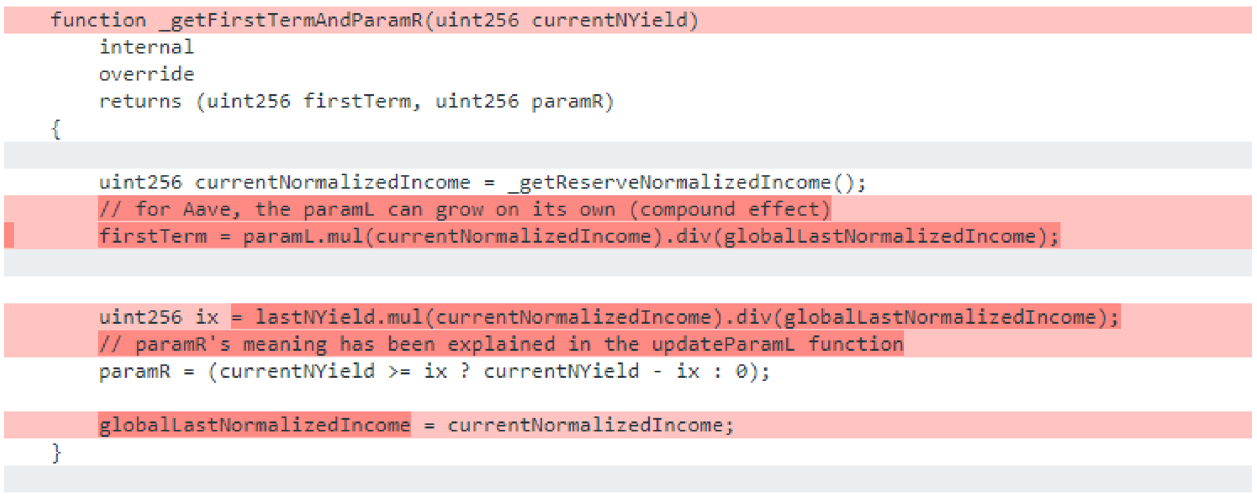
This issue has been resolved [here](#).

PEN-007: Inconsistency in _getFirstTermAndParamR between PendleAaveMarket and PendleAaveLiquidityMining

Severity: Low
Status: Resolved

For _getFirstTermAndParamR, the ix calculation in PendleAaveMarket uses mul and div, while the ix calculation in PendleAaveLiquidityMining uses rmul and rdiv.

Red:PendleAaveMarket



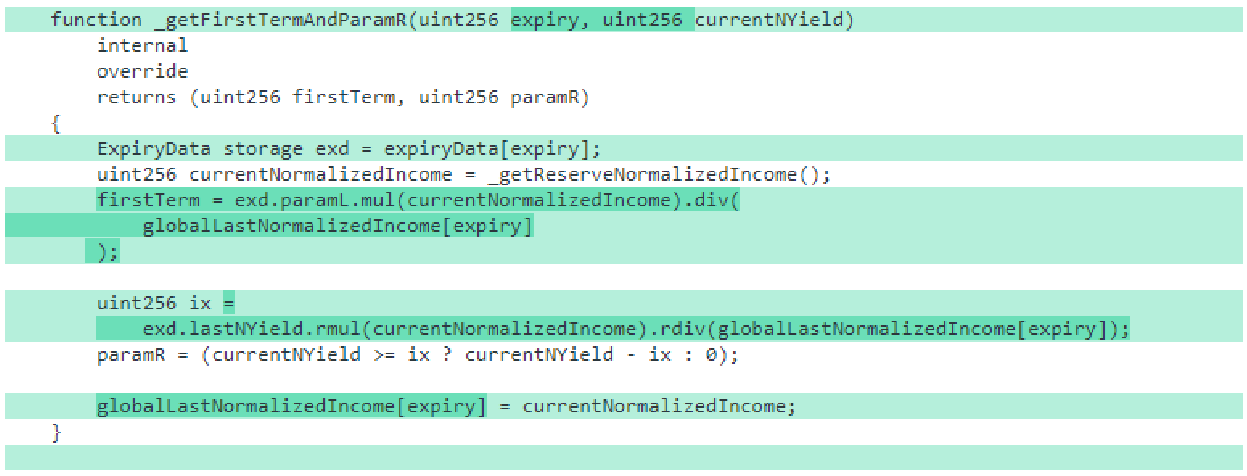
```
    function _getFirstTermAndParamR(uint256 currentNYield)
        internal
        override
        returns (uint256 firstTerm, uint256 paramR)
    {

        uint256 currentNormalizedIncome = _getReserveNormalizedIncome();
        // for Aave, the paramL can grow on its own (compound effect)
        firstTerm = paramL.mul(currentNormalizedIncome).div(globalLastNormalizedIncome);

        uint256 ix = lastNYield.mul(currentNormalizedIncome).div(globalLastNormalizedIncome);
        // paramR's meaning has been explained in the updateParamL function
        paramR = (currentNYield >= ix ? currentNYield - ix : 0);

        globalLastNormalizedIncome = currentNormalizedIncome;
    }
```

Green: PendleAaveLiquidityMining



```
    function _getFirstTermAndParamR(uint256 expiry, uint256 currentNYield)
        internal
        override
        returns (uint256 firstTerm, uint256 paramR)
    {
        ExpiryData storage exd = expiryData[expiry];
        uint256 currentNormalizedIncome = _getReserveNormalizedIncome();
        firstTerm = exd.paramL.mul(currentNormalizedIncome).div(
            globalLastNormalizedIncome[expiry]
        );

        uint256 ix =
            exd.lastNYield.rmul(currentNormalizedIncome).rdiv(globalLastNormalizedIncome[expiry]);
        paramR = (currentNYield >= ix ? currentNYield - ix : 0);

        globalLastNormalizedIncome[expiry] = currentNormalizedIncome;
    }
```

Recommendations
In terms of formula they are the same value, but in terms of precision .mul() then .div() is more precise. Modify the PendleAaveLiquidityMining contract to use mul() and div() instead.

This issue has been resolved here.

PEN-008: Unused modifiers in PendleRewardManager

Severity: Informational
Status: Resolved

OnlyForge and OnlyRouter modifiers are not used in PendleRewardManager, and can be removed.

Recommendations
Remove the unused modifiers.

This issue has been resolved here.

PEN-009: Tokens can be sent to their token contracts

Severity: Informational
Status: Resolved

In the internal transfer function of PENDLE token and PendleBaseToken, there is a lack of check allowing the destination address of transfers to be the contract itself.

Recommendations
It has been confirmed that there will be no use case where the token contract should receive its own token. A check such as the following can be added to prevent that from happening.

```
require(dst != address(this));
```

This issue has been resolved here.

Conclusion

All of the issues raised in the audit have been mitigated, and tests have also been added to the identified cases to ensure that the vulnerable scenarios no longer occur. Overall, the team was quick to respond to security issues raised and have implemented the security recommendations mentioned in the report.