Pendle Security Analysis

1 Overview Abstract

In this report, we consider the security of the Pendle project. Our main task is to find and describe security issues in the smart contracts of the platform to the team.

1.1 Limitations and use of the report

Broadly speaking, the assessments can not uncover all vulnerabilities of the given smart contracts, thus, it is not guaranteed that the system is secured even if no vulnerabilities are found. The focus of the assessments was limited to given smart contracts, other contracts were excluded (including external libraries or third party codes).

1.2 Summary

We have found **0** high severity issues, **0** medium severity issues and **5** low severity issues.

After the first iteration, the team has fixed all issues.

1.3 Recommendations

We recommend the team to fix all issues, as well as test coverage to ensure the security of the contracts.

2 Assessment Overview

2.1 Scope of the audit

Source codes for the audit was initially taken from the commit hash *d8713b12676339d68f2df5ec154c5cab21d9f526*. In-scope contracts for audit:

- All contracts except contracts that are in **misc** and **proxies** folders.

After the first iteration, team has fixed all issues and the final codebase was taken from the new commit hash *01154885472188022518745f2ca08104955b8b2b*

3 System Overview

The new codebase integrates the Pendle protocol with other protocols on Avalanche network like BenQi, Wonderland, Sushiswap.

The system overview is based on the provided documentation that is available on their website here, light paper can be found here.

4 Findings

We have found **0** high severity issues, **0** medium severity issues and **5** low severity issues.

After the first iteration, the team has fixed all issues.

1.1 [Fixed] [Low] PendleYieldTokenHolderBaseV2Multi: weth is unused

weth is unused and should be moved to **PendleBenQiYieldTokenHolder**

1.2 [Fixed] [Low] PendleRewardManagerMulti: comment mismatches with the implementation

Wrong comment for the **setSkippingRewards** function.

```
/**
Use:
    To set how often rewards should be updated for yieldTokenHolders of an underlyingAsset
Conditions:
    * The underlyingAsset must already exist in the forge
    * Must be called by governance
*/
function setSkippingRewards(bool _skippingRewards) external onlyGovernance {
    skippingRewards = _skippingRewards;
    emit SkippingRewardsSet(_skippingRewards);
}
```

1.3 [Fixed] [Low] PendleRewardManagerMulti: comment mismatches with the implementation

Wrong comment for the **redeemRewards** function as the logic is not for *COMP/StkAAVE*.

```
/**
Use:
    To claim the COMP/StkAAVE for any OT holder.
    Newly accrued rewards are equally accrued to all OT holders in the process.
Conditions:
    * Can be called by anyone, to claim for anyone
INVARIANTs:
    * this function must be called before any action that changes the OT balance of user
      * To ensure this, we call this function in the _beforeTokenTransfer hook of the OT token contract (indirectly through the forge)
*/
function redeemRewards(
    address _underlyingAsset,
    uint256 _expiry,
    address _user
)
    external
    override
    isValidOT(_underlyingAsset, _expiry)
    nonReentrant
    returns (TrioUints memory dueRewards)
```

1.4 [Fixed] [Low] Redundant return for PairTokensLib and TrioTokensLib

Return is redundant for function **allowance** in **PairTokenLib** and **TrioTokensLib**.

1.5 [Fixed] [Low] Break immutable property for reward tokens in PairTokensLib and TrioTokensLib

TrioTokensLib and **PairTokensLib** break the immutable property. Suggestion:

```
IERC20 internal immutable rewwardTokenA;
IERC20 internal immutable rewwardTokenB;

function rewardTokens() public returns (PairTokens) {
  return PairTokens {tokenA: rewwardTokenA, tokenB:rewwardTokenB }
}
```