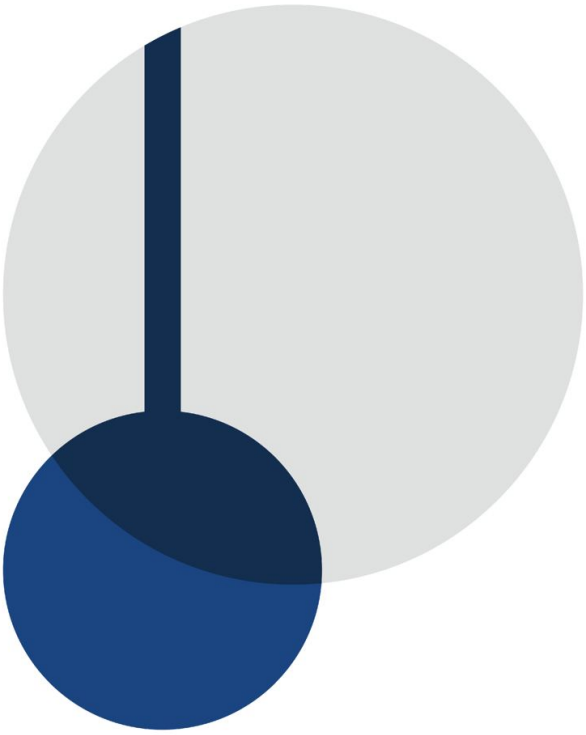

Pendle Security Analysis

This is a public report.

Published: June 14, 2021.

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.

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).

Audit Summary

We have found 1 medium severity issue and 7 low severity issues.

The identified issues have been addressed and not presented in the latest code.

Recommendations

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

Assessment Overview

Scope of the audit

Source codes for the audit was initially taken from the commit hash 48c039fb8b0089618f00113c65f26d9f3c4cfe00. In-scope contracts for audit:

- core: All contracts except contracts in **emergency** folder.
- interfaces: All contracts except for **IPendleGovernance.sol** and **IPendleTreasury.sol**.
- libraries: All contracts.
- periphery: All contracts.
- readers: All contracts.
- tokens: all contracts except for WETH9.sol.

Other source codes are out of scope for this report.

Update: After the initial report, the source code and scope of the audit have been updated to the commit hash 4499f36bd58736b1b0fc48f77ffed19f04ad9308

System Overview

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

1. Introduction

Pendle Finance leverages on the base lending layer created by prominent DeFi protocols such as Aave and Compound, which has shown incredible growth and community acceptance. The protocol builds on this layer by separating the future cash flows from these lending protocols' yield tokens and tokenizing it. This allows future yield to be traded without affecting ownership of the underlying asset.

2. Expectations on How It Works

2.1. Yield Tokenization

- For tokenization functions, the user solely interacts with the core routing contract **PendleRouter.sol**. The **PendleRouter.sol** contract routes to the relevant Forge contracts for tokenization.
- Users should be able to tokenize any Yield Bearing Token (such as *aTokens* or *cTokens*) as long as they are officially supported by the underlying yield protocol, namely Aave and Compound.
- The user can deposit and lock his Yield Bearing Token into a Forge, and the Forge mints Future Yield Tokens (XYTs) and Ownership Tokens (OTs), which is transferred back to the user.
- Users can indicate an expiry time for a future yield token.
- The user can redeem his accrued interest at any time given his XYT holdings.

- Whenever the Future Yield tokens are transferred to another owner, the interest of the yield token until current time is accrued to the curent owner
- The user can redeem the underlying Yield Bearing Token from the Forge as long as they provide the equal amounts of XYT and OT tokens to redeem the specified amount of Yield Bearing Token.
- Should the yield contract of the XYT token expire, the user has the choice to renew the yield contract, setting a new expiry, or to redeem the Yield Bearing Token by only returning back the OT tokens.
- Should the yield contract of the XYT token expire, the XYT tokens themselves will still exist but will be considered valueless.
- The protocol charges a fee for the interest that comes from XYT tokens.

2.2.Future Yield Trading via AMMs

- Users should be able to create a market between an ERC20 token and a XYT token. As proposed in the lite paper, Pendle's automated market making curve is designed to address the time sensitive nature of XYT. Users will need to only provide token allowance to the core routing contract, **PendleRouter.sol**, for trading.
- Pendle will enable permissionless creation of markets of all legitimate XYTs in the network.
- Users can add liquidity to any Market permissionlessly and get back an LP token as a representation of their stake.
- Users can add and remove liquidity for a Market by a single token or both tokens.
- Users can earn and redeem interest from the underlying protocol for as long as they retain the ownership of XYTs, which includes the use of XYTs for liquidity provision in Pendle AMM.
- Users can withdraw their liquidity at any point as long as they provide the specific LP token that corresponds to its intended Market to redeem their stake.
- Pendle charges a small swap fee for every transactions, modelled after Uniswap.

3. Protocol Overview

Ownership of the future cash flows is guaranteed by the smart contract, so there is no need to worry about collateral or counterparty risk, as long as the underlying lending protocol is not compromised.

3.1.PendleRouter.sol

The main entry point for users to perform tokenization operations and to perform trades across the different future yield markets. The PendleRouter contract itself acts as a router and communicates with other Pendle contracts.

3.2.PendleData.sol

Stores the data across forges and markets.

3.3.PendleForgeBase.sol and Pendle*Forge.sol

PendleForgeBase.sol implements the common logic for a forge, while the **Pendle*Forge.sol** contracts implements the specific implementation for AaveV2 and Compound

3.4.PendleMarketFactoryBase.sol and Pendle*MarketFactory.sol

Used to create new market contracts. **Pendle*MarketFactory.sol** are specific market factories for Aave and Compound, extending from the common **PendleMarketFactoryBase.sol**

3.5.PendleMarkets

Pendle AMM details can be found in Pendle AMM and Pendle AMM Design Paper.

Findings

We have found 1 medium severity issue and 7 low severity issues.

The identified issues have been addressed and not presented in the latest code.

Medium severity issues

The audit showed only 1 medium severity issue.

[Fixed] **PendleRouter.sol: strict requirement for msg.value when swapExactOut with source token is ETH or native token**

In PendleRouter, function _swapExactOut, after calculating the amount of source token that is needed to get the expected amount out, in case the source token is ETH or native token, it is stated that msg.value must be strictly equal the calculated source amount:

```
require(msg.value == transfer.amount, "ETH_SENT_MISMATCH");
```

From users/clients, it is too difficult to broadcast a transaction with the exact msg.value for the trade when using _swapExactOut function, thus, most of the trades with source token (either ETH or native token) will likely get reverted.

We recommend that the requirement should change to

```
require(msg.value >= transfer.amount);
```

and refund back users any ETH (or native token) that is not used for the swap.

Comment: This issue has been fixed and not presented in the latest code.

Low severity issues

The audit showed 7 low severity issues.

[Fixed] **PendleAaveMarket.sol: possible underflow for _getIncomeIndexIncreaseRate**

In **PendleAaveMarket**, function **_getIncomeIndexIncreaseRate**:

```
return _getReserveNormalizedIncome().rdiv(globalLastNormalizedIncome) - Math.RONE;
```

It could lead to underflow and return unexpected data, consider using SafeMath here.

Comment: This issue has been fixed and not presented in the latest code.

[Mitigated] **PendleLiquidityMiningBase: EIP 3074 breaks the check msg.sender == tx.origin for EOAs.**

The modifier nonContractOrWhitelisted uses msg.sender == tx.origin to check whether the sender is the EOA or a contract. This can be bypassed by an attacker who colludes with a miner after EIP-3074 is live.

```
msg.sender == tx.origin || whitelist.whitelisted(msg.sender)
```

Consider using isContract check from OpenZeppelin which is more common.

*Comment: The team has mitigated the issue by adding the **isContract** check.*

[Fixed] **PendleLiquidityMiningBase.sol: _allowedToWithdraw breaks some assumptions**

After adding _allowedToWithdraw function, it breaks the assumption that there is always enough Pendle token in the contract. Thus, the amount of PENDLE token in the contract needs to be carefully reviewed after the addition of new features.

Comment: This issue has been fixed and not presented in the latest code. The Governance is not allowed to withdraw the PENDLE token.

[Fixed] **PendleForgeBase.sol: redundant checks for enough balance before burning**

In PendleForgeBase.sol, it is redundant to check for enough balance in lines 248, 249 before calling burn function because the burn function will revert if balance is not enough.

```
require(tokens.ot.balanceOf(_user) >= _amountToRedeem);
require(tokens.xyt.balanceOf(_user) >= _amountToRedeem);
```

Comment: This issue has been fixed and not presented in the latest code.

[Fixed] **PendleForgeBase.sol: Inaccurate comment at line 175**

In PendleForgeBase, line 175, it is stated that the “ot address is passed in to be used in th salt of CREATE2”, however, the logic doesn’t use CREATE2 anymore.

```
// ot address is passed in to be used in the salt of CREATE2
yieldTokenHolders[_underlyingAsset][_expiry] = ……..
```

Comment: This issue has been fixed and not presented in the latest code.

[Fixed] **PendleWhitelist.sol: Consider using AddressSet for whitelisted to allow reading the list of whitelisted addresses**

The current implementation in the PendleWhitelist only allows to check if an address has been whitelisted, but can not read the list of whitelisted addresses, thus, it is difficult for add/remove operations.

Consider using AddressSet for whitelisted.

Comment: This issue has been fixed and not presented in the latest code. However, the getWhitelist() function could be out of gas if there are too many whitelisted addresses (won't likely happen), it will be nice to have a function to get a whitelist address at a specified index.

[Fixed & Acknowledged] **Other recommendations**

PendleRouter.sol

i) Line 34: redundant import IPendleRewardManager.

ii) Line 77: receive function should use require instead of assert to avoid burning all remaining gas.

iii) Line 649: Unused function _getMarketToken.

Comment: All issues have been fixed and not presented in the latest code.

IPendleWhitelist.sol

i) All imports are redundant.

Comment: All issues have been fixed and not presented in the latest code.

PendleMarketReader.sol

i) Line 46: data should be immutable.

ii) Should import MarketMath to reuse the logic.

Comment: (i) is acknowledged, (ii) has been fixed.

PendleForgeBase.sol, **PendleMarketBase.sol**, **PendleLiquidityMiningBase.sol**

i) checkNotPaused should be modifier to help reduce gas consumption, however, it will increase byte code size of the contract.

Comment: Acknowledged. Due to the byte code size of these contracts, the team decided to keep using function instead of modifier.

MathLib.sol

i) Consider using formula from Balancer to calculate the e^x (0 <= x <= 1).

ii) Consider updating to the latest math from Balancer v2 here.

iii) Consider removing checkMultOverflow and use SafeMath instead.

iv) Function rpowe states that "if the function cannot converge, it will lead to our of gas", however, the function will revert when it reaches 500 iterations, thus, it may not be out of gas at that point as commented, also consider exiting the loop or revert by require.

Comment: (i), (ii), (iii) are acknowledged, all maths have been well-tested, (iv) has been fixed.

PendlePausingManager.sol

i) Consider moving permForgeHandlerLocked, permMarketHandlerLocked, permLiqMiningHandlerLocked into EmergencyHandlerSetting and make a library for it to reduce duplicated codes.

Comment: Acknowledged.

PendleMarketBase.sol

i) After removing liquidity, users will have to make some calls again to claim any due interests.

ii) Function _updateDueInterests: exd.paramL and exd.lastParamL[user] should be cache in memory for gas savings.

Comment: (i) is acknowledged, (ii) has been fixed.

PendleData.sol

i) markets[key] seems to be redundant, using only getMarket[_marketFactoryId][_xyt][_token] should be enough, thus, can reduce some gas consumptions for addMarket function.

Comment: Acknowledged.

General: Should reorganize the interfaces folder.

Comment: Acknowledged.

Testing

The team has provided tests for all contracts, they do not provide 100% full coverage yet. We strongly recommend the team to add full test coverage to ensure the security of the codes.