Custodia Security

Noya Review

Conducted By: Ali Kalout, Ali Shehab

Contents

1. Disclaimer

A smart contract security review cannot ensure the absolute absence of vulnerabilities. This process is limited by time, resources, and expertise and aims to identify as many vulnerabilities as possible. We cannot guarantee complete security after the review, nor can we assure that the review will detect every issue in your smart contracts. We strongly recommend follow-up security reviews, bug bounty programs, and on-chain monitoring.

2. Introduction

Custodia conducted a security assessment of Noya’s Pendle Connector smart contract ensuring its proper implementation.

3. About Noya

NOYA represents a paradigm shift in decentralized finance, introducing a protocol that empowers AI agents to control liquidity across multiple chains with unparalleled trustlessness and precision. Engineered with a foundational composable system, NOYA built from the ground up a secure private keeper network, a trustless AI-compatible oracle, and a competitive environment for AI architects alongside strategy managers.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1. Impact

- High: Results in a substantial loss of assets within the protocol or significantly impacts a group of users.
- Medium: Causes a minor loss of funds (such as value leakage) or affects a core functionality of the protocol.
- Low: Leads to any unexpected behavior in some of the protocol's functionalities, but is not critical.

4.2. Likelihood

- High: The attack path is feasible with reasonable assumptions that replicate on-chain conditions, and the cost of the attack is relatively low compared to the potential funds that can be stolen or lost.
- Medium: The attack vector is conditionally incentivized but still relatively likely.
- Low: The attack requires too many or highly unlikely assumptions, or it demands a significant stake by the attacker with little or no incentive.

4.3. Action required for severity levels

- Critical: Must fix as soon as possible
- High: Must fix
- Medium: Should fix
- Low: Could fix

5. Security Assessment Summary

Duration: 26/04/2025
Repository: Noya-ai/noya-vault-contracts
Commit: 66eb63b87093a01b7a8837b4b279fcee300adf8b

- contracts/connectors/PendleConnector.sol

6. Executive Summary

Throughout the security review, Ali Kalout and Ali Shehab engaged with Noya to review Noya. In this period a total of 2 issues were uncovered.

Findings Count

Severity	**Amount**
Critical	N/A
High	1
Medium	N/A
Low	1
Total Finding	**2**

Summary of Findings

ID	Title	Severity	Status
H-01	`_getPositionTVL` reverts if the Pendle market is expired, leading to full DoS	High	Resolved
L-01	Tokens in registry are not updated after `supply`	Low	Resolved

7. Findings

7.1. High Findings

[H-01] `_getPositionTVL` reverts if the Pendle market is expired, leading to full DoS

Severity:
High

Description:
The `PendleConnector::_getPositionTVL` function does not properly account for Pendle market expiry. When a market expires, Yield Tokens (YT) immediately lose all value (their redeemable value becomes zero), and functions like `getLpToAssetRate` and `getPtToAssetRate` may revert if called. Since `_getPositionTVL` still tries to read LP and PT rates after expiry without checking, this causes a full revert. As a result, any vault holding an expired Pendle position will experience a complete denial-of-service (DoS) when trying to calculate TVL, blocking withdrawals, rebalancing, and other dependent operations.

Proof of Concept:
```
function test_DoSAfterExpiry() public {
    chainlinkOracle.updateDefaultChainlinkPriceAgeThreshold(10 days - 1);
    uint256 amount = 100e6;

    _dealERC20(USDT, address(connector), amount);

    vm.startPrank(owner);
    connector.updateTokenInRegistry(address(USDT));

    connector.supply(pendleUsdtMarket, amount, 1);

    connector.mintPTAndYT(pendleUsdtMarket, 10e6);

    vm.warp(block.timestamp + IPMarket(pendleUsdtMarket).expiry());

    assertTrue(IPMarket(pendleUsdtMarket).isExpired());

    vm.expectRevert();
    accountingManager.TVL();
}
```

Recommendations:

Refactor `PendleConnector::_getPositionTVL` to account for the expired market:

```
function _getPositionTVL(HoldingPI memory p, address base) public view override returns
(uint256 tvl) {
    PositionBP memory positionInfo = registry.getPositionBP(
        vaultId,
        p.positionId
    );
    if (positionInfo.positionTypeId == PENDLE_POSITION_ID) {
        uint256 underlyingBalance;
        address market = abi.decode(positionInfo.data, (address));
        (
            IPStandardizedYield _SY,
            IPPrincipalToken _PT,
            IPYieldToken _YT
        ) = IPMarket(market).readTokens();
        (, address _underlyingToken, ) = _SY.assetInfo();
        bool isExpired = block.timestamp > IPMarket(market).expiry();
        if (isExpired) {
            // After expiry: only count SY
            uint256 SYAmount = _SY.balanceOf(address(this));
            // Burn LP manually if you want before calling TVL function
            // PT must be redeemed manually
            underlyingBalance = (SYAmount * _SY.exchangeRate()) / 1e18;
        } else {
            // Before expiry: normal behavior
            uint256 lpBalance = IERC20(market).balanceOf(address(this));
            if (lpBalance > 0) {
                underlyingBalance +=
                    (lpBalance * IPMarket(market).getLpToAssetRate(10)) /
                    1e18;
            }
            uint256 PTAmount = _PT.balanceOf(address(this));
            if (PTAmount > 0)
                underlyingBalance +=
                    (PTAmount * IPMarket(market).getPtToAssetRate(10)) /
                    1e18;
            uint256 SYAmount = _SY.balanceOf(address(this));
            uint256 YTBalance = _YT.balanceOf(address(this));
            if (YTBalance > 0) SYAmount += getYTValue(market, YTBalance);
            if (SYAmount > 0)
                underlyingBalance += (SYAmount * _SY.exchangeRate()) / 1e18;
        }
        tvl = valueOracle.getValue(
            _underlyingToken,
            base,
            underlyingBalance
        );
    }
    return tvl;
}
```

7.2. Low Findings

[L-01] Tokens in registry are not updated after `supply`

Severity:
Low

Description:
`PendleConnector::supply` doesn't update the underlying token in the registry, to remove it from the connector's position in case the remaining balance is below the DUST level.

Recommendations:
Add `_updateTokenInRegistry(_underlyingToken)` just after the deposit execution.