Siloed Pinto

Security review

Version 1.0

Reviewed by
nmirchev8
deth

Table of Contents

1 About Egis Security

Egis Security is a team of experienced smart contract researchers, who strive to provide the best smart contract security services possible to DeFi protocols.

The team has a proven track record on public auditing platforms like Code4rena, Sherlock, and Cantina, earning top placements and rewards exceeding $170,000. They have identified over 150 high and medium-severity vulnerabilities in both public contests and private audits.

2 About Siloed Pinto

Siloed Pinto is an ERC-4626 yield-bearing vault designed to optimize returns for Pinto asset holders. Users can deposit Pinto into the contract, which then automatically allocates the funds into Silo, a yield-generating protocol from Pinto ecosystem on Base network.

Key Features:

- Reward Vesting Mechanism → When rewards are collected, they are vested back into the vault to mitigate just-in-time liquidty attacks.
- Automated Yield Generation → Deposited Pinto accrues yield through Silo, optimizing returns.

Siloed Pinto enhances ERC-4626 with native Silo integration, offering users greater flexibility in managing their deposits while ensuring efficient, automated, and structured yield generation.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

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** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	Siloed Pinto
Repository	Private
Commit hash	2fa16cd6a4047224ab82e14996ec097b2e66bbc5
Resolution	1df51f88813c4dcd57fc05a2d84386f435df2201
Documentation	-
Methods	Manual review

Scope

src/AbstractSiloedPinto.sol
src/SiloedPinto.sol
src/LibPrice.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	1
Low risk	3
Informational	2

6 Findings

6.1 Medium risk

6.1.1 Someone can front-run user that wants to redeemToSilo by creating new deposits

Severity: *Medium risk*

Context: SiloedPinto.sol#L292-L302

Description:

If last deposits are of stems with grown stalks and `SiloedPinto` user wants to benefit from that and call `redeemToSilo`, another user can brick his intentions by opening two new deposits, which will result in pushing new stems in `deposits` for germinating deposits. The problem may arise if two users submit transaction in the same time and the `deposit` is executed before `redeemToSilo`.
As a result,the victim receives a silo deposit, which is in germinating state and less favorable than the one that he has expected

Recommendation: Consider implementing `expectedStems` var to `redeemToSilo` and compare it's values against the stems returned from `_accountForOutboundDeposits`

Resolution: Acknowledged

6.2 Low risk

6.2.1 previewRedeemToSiloDeposits may return wrong data

Severity: *Low risk*

Context: SiloedPinto.sol#L429-L432

Description: If earnedPintoStem calculated in the function does not match any of the existing stems in deposits and we withdraw from stem that is < earnedPintoStem the func won't enter else if (modified_stems[i] == earnedPintoStem) branch, which assigns the shifting to the modified_amounts:

```
            else if (modified_stems[i] == earnedPintoStem) {
                modified_amounts[i] = amounts[i] + shifting;
                shifting = 0;
            }
```

This will result in returning modified_amounts with smaller value than expected

The problem comes from the fact that in previewRedeemToSiloDeposits we don't _claim and we don't have a guarantee that we have earnedPintoStem in the deposit array. While in redeemToSilo it is safe because upOrRight calls _claim, which will insert the corresponding stem, if it needed.

Recommendation: In the else check whether shifting has succesfully been assigned

```
            else {
                // if shifting > 0 => we should introduce ‘earnedPintoStem‘ to
                    modified_amounts
                modified_amounts[i] = amounts[i];
            }
```

Resolution: Fixed by removing the func

6.2.2 Consider making SLIPPAGE_RATIO configurable

Severity: *Low risk*

Context: AbstractSiloedPinto.sol#L38-L39

Description: In _floodSwapWell we check if current well reserves are within 1% slippage of the EMA price, so we continue with the flow. 1% is a low value for highly volatile wells such as PINTO/WETH.

Recommendation: Consider configuring the variable to prevent blocking the funds in the contract for large periods.

Resolution: Fixed

6.2.3 Consider implementing a __gap in AbstractSiloedPinto

Severity: *Low risk*

Context: AbstractSiloedPinto.sol#L14

Description: For upgradeable contracts, there must be storage gap to “allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments”. Otherwise it may be very difficult to write new implementation code. Without storage gap, the variable in child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious consequences to the child contracts.

Reference from Open Zeppelin

Recommendation: Add a `__gap` at the bottom of the contract.

Resolution: Fixed

6.3 Informational

6.3.1 Remove `console.logs` in `LibPrice.sol`

Severity: *Informational*

Resolution: Fixed

6.3.2 `_checkMaxDeposit` and `_checkMaxMint` checks are redundant

Severity: *Informational*

Description: The checks are redundant because max possible value for assets is equal to the max possible unit number `(type(uint256).max)`

Resolution: Acknowledged