Siloed Pinto
Security Review

Cantina Managed review by:

Om Parikh, Security Researcher
T1moh, Associate Security Researcher

February 27, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Pinto is low volatility money built on Base. Pinto's primary objective is to incentivize independent market participants to regularly cross the price of 1 Pinto over its 1 Dollar peg in a sustainable fashion.

From Feb 6th to Feb 14th the Cantina team conducted a review of siloed-pinto on commit hash 2fa16cd6. The team identified a total of **10** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	0	1
Low Risk	7	6	1
Gas Optimizations	0	0	0
Informational	2	2	0
Total	**10**	**8**	**2**

3 Findings

3.1 Medium Risk

3.1.1 Future depositors receive a portion of the reward that began to be distributed before their deposit

Severity: Medium Risk

Context: SiloedPinto.sol#L310-L312

Description: `sPinto` is an ERC4626 yield bearing token. To avoid value extraction via flashloans, it implements vesting mechanism. Once the reward is received, it is distributed within the next 2 hours.

However deposits which happen after vesting start will still receive part of the rewards, consider following example:

1. Total assets and shares is 100. 50 assets are now vested.
2. After 1 hour user deposits 125 assets, he receives `125 * 100 / 125 = 100` shares. That's because half of vested rewards is unlocked.
3. After one hour user redeems 100 shares for `100 * 275 / 200 = 137.5` assets.

So when there is big enough part of vested rewards, it becomes profitable to make a short-term deposit.

Recommendation: Mitigation is not so obvious. One of solutions can be to:

1. Calculate `totalAssets()` as in current implementation when user redeems/withdraws.
2. However calculate `totalAssets()` as `underlyingPdv` on mint/deposit, i.e. don't subtract unvested rewards.

It makes user to receive slightly less shares on deposit, so that it's not profitable to execute the attack. When user deposits, waits 2 hours, withdraws - in this case he doesn't lose value.

However user loses when makes withdrawal in less then 2 hours after deposit. Loss is equal to unvested reward, so that described attack is not profitable.

PintoFarm: A user could deposit after yield has been earned and withdraw as soon as the yield has dried up. when they do this they dilute holders of the token including this specific case - that they are receiving yield from cycles before they contributed to the token. This is not a perfectly efficient mechanism, because yield is not necessarily distributed in perfect order and proportion across time. However, the risk is mitigated because the value extractor would need to keep their capital in the token across the vesting period (currently set to multiple hours), which exposes them to the same risk as holding the token for the extra minute it would have taken to "earn" the interest. The vesting period can be extended to further discourage this behavior. It may be viable to extend it to many days.

Cantina Managed: Acknowledged.

3.2 Low Risk

3.2.1 `previewRedeemToSiloDeposits()` incorrectly accounts deposit claimed from `earnedBeans`

Severity: Low Risk

Context: SiloedPinto.sol#L414-L419

Description: Function `previewRedeemToSiloDeposits()` imitates redeem and should return an array of SIlo deposits which will be withdrawn. It should also account the claiming of newly deposited `earnedPinto` before withdrawing deposits.

Let's observe it by example:

1) Suppose user wants to withdraw 100 Pinto.
2) There are 2 deposits with 100 Pinto each, so initial array would be `[100]` - i.e. withdraw 1 full deposit.
3) However `earnedPinto` is 50 Pinto. It means that before withdrawing there will be new deposit of 50 Pinto with the latest Stem. It means that actual array of withdrawn deposits is `[50, 50]`.

Problem is that actual implementation calculates [50, 0]. It doesn't fill the latest value in array.

That's because `modified_amounts` has bigger length:

```
int96[] memory modified_stems = new int96[](stems.length + 1);
uint256[] memory modified_amounts = new uint256[](amounts.length + 1); // <<<
// uint256 shiftRemaining = earnedPinto;
uint256 shifting;

for (i = 0; i < stems.length; i++) { // <<<
    modified_stems[i] = stems[i];
    // If before claim stem, reduce amount.
    if (modified_stems[i] < earnedPintoStem) {
        // ...
    }
    // Add the shifted Pinto to the claim stem.
    else if (modified_stems[i] == earnedPintoStem) {
        // ...
    }
    // Do not modify higher stem deposits.
    else {
        // ...
    }
}
```

Proof of Concept: Insert this test into `test/unit/SiloedPintoFromDepositsTest.sol`:

```
function test_custom() public {
    uint256 assets = 100e6;
    // deposit and mint sPinto to receiver external balance
    uint256 shares1 = sPinto.depositAdvanced(assets, receiver, From.EXTERNAL, To.EXTERNAL);
    // call sunrise to get new stems
    proceedAndGm();
    uint256 shares2 = sPinto.depositAdvanced(assets, receiver, From.EXTERNAL, To.EXTERNAL);
    // deposit list [100 100]

    // End germination for those deposits
    proceedAndGm();
    proceedAndGm();
    proceedAndGm();


    // get the length of the deposits pre withdraw
    uint256 preLength = sPinto.getDepositsLength();
    assertEq(preLength, 2);

    vm.mockCall(PINTO_PROTOCOL, abi.encodeWithSignature("balanceOfEarnedBeans(address)", address(sPinto)),
    ↪   abi.encode(uint256(50e6)));

    (, uint256[] memory amounts) = sPinto.previewRedeemToSiloDeposits(100e18);

    // Prints [50e6, 0]
    for (uint256 i; i < amounts.length; i++) {
        console.log(amounts[i]);
    }
}
```

Recommendation: Loop through `modified_stems.length` and handle last case.

PintoFarm: Upon further thought `previewRedeemToSiloDeposits` will be removed. People redeeming/withdrawing to silo deposits should expect to receive the worst possible deposits that correspond to the pdv of their shares. Meaning that the deposits they receive can be germinating and/or have 0 grown stalk. In this context, previewing the deposits and introducing so much complexity in the code makes little sense. Addressed in PR 18.

Cantina Managed: Fix verified.

3.2.2 `_unvestedAssets` rounds down irrespective of how it is used for further calculations

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In `_unvestedAssets`:

```
return (timeRemaining * vestingPinto) / VESTING_PERIOD;
```

This calculation rounds down the return value. It is further used in `totalAssets` which is then used preivew and convert ERC4626 functions. since the value is already truncated, openzeppelin rounding direction will still not yield the accurate output.

Recommendation: rounding direction in `_unvestedAssets` should be input basis use case.

PintoFarm: When we round down here, we round down totalAssets. Essentially that still benefits the contract when redeeming/depositing which is what we want since no value can be leaked.

Cantina Managed: Acknowledged.

3.2.3 Missing gaps in upgradeable contracts with inheritance hierarchy

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: `SiloedPinto` contract and its parent contract `AbstractSiloedPinto` are upgradeable contracts, they storage significant amount of state and also the external protocol integrations are upgradeable. However, contracts are missing gaps to safely extend storage with future upgrades.

Recommendation: Add storage gaps in `SiloedPinto` or `AbstractSiloedPinto`.

PintoFarm: Addressed in PR 15.

Cantina Managed: Fix verified.

3.2.4 `rescueTokens` can be used to withdraw any token contrary to its documented usage

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description:

```
/**
 * @notice Allows the owner to rescue tokens. Serves 2 purposes:
 * - Recovers various tokens accidentally sent to the contract.
 * - Recovers flood assets in case of emergency/bug in flood logic.
 * @param token The token to be rescued.
 * @param amount The amount of tokens to be rescued.
 * @param to Where to send rescued tokens
 */
function rescueTokens(address token, uint256 amount, address to) external nonReentrant onlyOwner {
    // @audit disallow certain tokens (address(this))
    IERC20(token).safeTransfer(to, amount);
}
```

Apart from use cases mentioned in nat-spec above, this function can be used to withdraw `address(this)` (`sPinto` vault shares) and external `pinto` tokens.

Recommendation: this behaviour should be either documented or prevented.

PintoFarm: Added more natspec in PR 20.

Cantina Managed: Fix verified.

3.2.5 `VESTING_PERIOD` is hardcoded while it can be changed upstream

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description:

```
uint256 constant VESTING_PERIOD = 2 hours;
```

`VESTING_PERIOD` is used to determine the amount of pinto vested and remaining. however this can be changed in upstream protocol and staked pinto vaults will outdated value resulting in wrong calculations.

Recommendation: Query it from the the upstream pinto core protocol or atleast make this configurable.

PintoFarm: We decided to make this configurable along with some other constants in PR 17.

Cantina Managed: Fix verified.

3.2.6 Use dynamic slippage value instead of hardcoded `SLIPPAGE_RATIO`

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description:

```
uint256 constant SLIPPAGE_RATIO = 0.01e18; // 1%
```

Due to using fixed constant value:

- Protocol will leak value via MEV during swaps and flood.
- Will prevent users from depositing / withdrawing (temporary DOS) if slippage is high.

Recommendation: consider making it configurable parameter and monitor it offchain to set accordingly.

PintoFarm: We decided to make this configurable along with some other constants in PR 17.

Cantina Managed: Fix verified.

3.2.7 Implications of using reserves for deriving price on-chain

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `LibPrice` queries `readCappedReserves` and `readInstantaneousReserves` from a pump (oracle) for deriving price, this price is further used to check slippage bounds. However, it has some shortcomings incase of a pair is relatively illiquid:

- Oracle might consume stale ema reserves, resulting to different price compared to price of that token in different well pair or on exchanges.
- If there have no swaps for a while (i.e stale price), it is relatively easier to manipulate ema on chains with faster block time, since last reserves are mapped `block.timestamp` which will update more frequently.
- There can be concerns of atomic sandwiching by a large LP in well who can abuse by removing liquidity from that pair to increase slippage without affecting ema by a wider margin.

Recommendation:

- Add a staleness check, consumed reserves should be not stale than a given threshold.
- Add adverserial test cases to obeserve impacts of removing liqudity and then swapping.
- Try to use off-chain oracles in some capacity if possible.

PintoFarm: We added a sync call to the well before calculating the price from the reserves to update the pump to its current reserves and avoid staleness. Also added a test in PR 21.

Cantina Managed: Partially fixed.

3.3 Informational

3.3.1 Use `PINTO_PROTOCOL.withdrawDeposits()` whenever it's possible

Severity: Informational

Context: AbstractSiloedPinto.sol#L183

Description: Currently you loop over every deposit and make a single transfer. However Pinto protocol has special function to transfer a batch of deposits called `wirhdrawDeposits()`.

Recommendation:

```
- for (uint256 i = 0; i < stems.length; i++) {
-     PINTO_PROTOCOL.withdrawDeposit(PINTO_ADDRESS, stems[i], amounts[i], To.INTERNAL);
- }
+ PINTO_PROTOCOL.withdrawDeposits(PINTO_ADDRESS, stems, amounts, To.INTERNAL);
```

PintoFarm: Addressed in PR 19.

Cantina Managed: Fix verified.

3.3.2 Remove all `console.log()` for production readiness

Severity: Informational

Context: LibPrice.sol#L43-L51, LibPrice.sol#L88, LibPrice.sol#L103, LibPrice.sol#L109-L115

Description: There are multiple `console.log()` in LibPrice.sol. It's a best practice to not use it in production ready code.

Recommendation: Consider removing it before deploying contracts.

PintoFarm: Addressed in PR 16.

Cantina Managed: Fix verified.