Three Sigma

# Code Audit

Ojo Smart Pendle

# Disclaimer

Code Audit

**Ojo** Smart Pendle

# Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

# Table of Contents

Code Audit

**Ojo** Smart Pendle

Table of Contents

# Summary

Code Audit

**Ojo** Smart Pendle

# Summary

Three Sigma audited Ojo Smart Pendle in a 0.2 person week engagement. The audit was conducted on 14/03/2025.

## Protocol Description

Ojo Network's Smart Oracle enhances price reliability for derivative assets by combining a primary market price feed, a secondary discount rate feed, and a decision mechanism that selects the lower value. This approach reduces reliance on hard-coded prices, mitigates market manipulation risks, and enables safer pricing for assets such as Pendle and Spectra PT tokens. By providing more robust price protection, Smart Oracles support risk-aware lending and improve capital efficiency in DeFi markets.

# Scope

Code Audit

**Ojo** Smart Pendle

# Scope

| Filepath | nSLOC |
| --- | --- |
| src/OjoPTFeed.sol | 58 |
| **Total** | **58** |

## Assumptions

No assumptions were made.

# Methodology

Code Audit

**Ojo** Smart Pendle

# Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

## Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

| Severity / Likelihood | LOW | MEDIUM | HIGH |
|---|---|---|---|
| NONE | None | | |
| LOW | Low | | |
| MEDIUM | Low | Medium | Medium |
| HIGH | Medium | High | High |
| CRITICAL | High | Critical | Critical |

# Project Dashboard

## Code Audit

**Ojo** Smart Pendle

# Project Dashboard

## Application Summary

| | |
|---|---|
| Name | Smart Pendle |
| Commit | 4dfe4a5 |
| Language | Solidity |
| Platform | Ethereum |

## Engagement Summary

| | |
|---|---|
| Timeline | 14/03/2025 to 14/03/2025 |
| № of Auditors | 1 |
| Review Time | 0.2 person weeks |

## Vulnerability Summary

| Issue Classification | Found | Addressed | Acknowledged |
|---|---|---|---|
| Critical | 1 | 1 | 0 |
| High | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 |
| Low | 0 | 0 | 0 |

| | | | |
|---|---|---|---|
| None | 4 | 4 | 0 |

## Category Breakdown

| | |
|---|---|
| Suggestion | 4 |
| Documentation | 0 |
| Bug | 1 |
| Optimization | 0 |
| Good Code Practices | 0 |

# Findings

## Code Audit

**Ojo** Smart Pendle

# Findings

## 3S-Ojo-C01

Lack of initialization status check in **`initialize`** function

| | |
|---|---|
| Id | 3S-Ojo-C01 |
| Classification | Critical |
| Severity | Critical |
| Likelihood | High |
| Category | Bug |
| Status | Addressed in #25e92ca. |

### Description

The **`OjoPTFeed`** contract includes an **`initialize`** function, which is intended to set up the contract's oracle feeds, **`FEED_1`** and **`FEED_2`**, after being deployed using OpenZeppelin's clone functionality. This function is important for the contract's operation as it assigns the addresses of the oracle feeds that the contract will use to fetch data. However, the **`initialize`** function currently lacks any mechanism to check whether it has already been called, allowing it to be invoked multiple times by any entity. This vulnerability could be exploited by an attacker to replace the legitimate oracle feeds with malicious contracts, potentially leading to the manipulation of data and the draining of any protocol integrating **`OjoPTFeed`**.

### Recommendation

It is suggested to implement a mechanism to ensure that the **`initialize`** function can only be called once. This can be achieved by checking the value of one of the two feeds, which can be zero only when initialized.

# 3S-Ojo-N01

Lack of staleness check in **`latestRoundData`** function

| | |
|---|---|
| Id | 3S-Ojo-N01 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in #25e92ca. |

## Description

The **`latestRoundData`** function in the **`OjoPTFeed`** contract is designed to return the latest data from two underlying oracles, **`FEED_1`** and **`FEED_2`**. It compares the answers from both feeds and returns the data from the feed with the lower value. However, the function currently lacks a staleness check, which means it does not verify whether the data from the oracles is up-to-date. This could lead to situations where outdated data is returned if one of the feeds has not been updated recently. The absence of a staleness check could be problematic.

## Recommendation

The desired behavior depends on how the entities integrating **`OjoPTFeed`** want to handle this kind of situation, for example they might require both of the oracles being fresh (not stale), which would not require any additional check on **`OjoPTFeed`**. The client is thus suggested to engage with the interested parties to understand what the behavior should be here. If the client opts for implementing a staleness check, it is suggested to skip it for oracles returning 0 as **`updatedAt`**.

# 3S-Ojo-N02

Variable shadowing in **`getRoundData`** function

| | |
|---|---|
| Id | 3S-Ojo-N02 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in #25e92ca. |

## Description

In the **`getRoundData`** function of the **`OjoPTFeed`** contract, there is an issue of variable shadowing that results in unnecessary gas overhead. Within this function, the variables **`roundId`**, **`answer`**, **`startedAt`**, **`updatedAt`**, and **`answeredInRound`** are declared twice: once in the function signature and again within the function body. This redundancy leads to shadowing, where the inner declarations override the outer ones, causing additional gas consumption without affecting the intended functionality of the contract.

## Recommendation

To optimize gas usage and improve code clarity, remove the redundant variable declarations within the **`getRoundData`** function body.

# 3S-Ojo-N03

**`getRoundData`** function reverts due to unknown round id

| | |
|---|---|
| Id | 3S-Ojo-N03 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in #25e92ca. |

## Description

The **`getRoundData`** function in the **`OjoPTFeed`** contract is designed to return data for a specific round identified by the **`_roundId`** parameter. However, the function currently reverts if the **`_roundId`** provided by the caller does not match the **`roundId`** returned by the **`latestRoundData`** function. The **`latestRoundData`** function selects data from one of two oracles, specifically the one with the lowest value. This behavior makes it impossible for external callers to predict the correct **`_roundId`** to use when calling **`getRoundData`**, as they cannot determine in advance which oracle's data will be returned. This limitation hinders the integration of the **`OjoPTFeed`** contract with external systems that rely on specific round data.

## Recommendation

To address this issue, introduce a new function that returns the currently active oracle, which is the one providing the lowest value among the two. This function will allow external contracts to determine which oracle is currently active and thus predict the correct **`_roundId`** to use when calling **`getRoundData`**.

# 3S-Ojo-N04

Inefficient gas usage in **`initialize`** function

| | |
|---|---|
| Id | 3S-Ojo-N04 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in #25e92ca. |

## Description

The **`initialize`** function in the **`OjoPTFeed`** contract is designed to set up the initial state of the contract by assigning the addresses of two feeds, **`FEED_1`** and **`FEED_2`**, and ensuring they have matching decimal values. However, the current implementation writes to storage before validating that the decimal values of the two feeds are equal. This approach is inefficient as it wastes gas if the validation check does not pass. The validation check should be performed using the function parameters instead of the more expensive storage variables.

## Recommendation

To improve gas efficiency, move the validation check for matching decimals before the storage assignments. Use the function parameters directly to perform this check.