THREE SIGMA

Ojo Yield Risk Engine

Security Review

Disclaimer

Security Review

Ojo Yield Risk Engine

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

Security Review

Ojo Yield Risk Engine

Table of Contents

Summary

Security Review

Ojo Yield Risk Engine

Summary

Three Sigma audited Ojo's Yield Risk Engine in a 0.4 person week engagement. The audit was conducted on 29/05/2025.

Protocol Description

YieldRiskEngine is a Chainlink-compatible smart contract factory that helps Morpho risk curators cap the price of yield-bearing tokens based on their expected yield. By preventing oracle overvaluation, it safeguards markets from positive-price-swing attacks and reduces the risk of bad debt from inflated collateral.

Scope

Security Review

Ojo Yield Risk Engine

Scope

Filepath	nSLOC
src/OjoYieldRiskEngine.sol	60
src/OjoYieldRiskEngineFactory.sol	67
Total	**127**

Methodology

Security Review

Ojo Yield Risk Engine

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 classify findings based on Immunefi's Vulnerability Severity Classification System (v2.3) as a guideline. The final classification considers both the potential impact of an issue, as defined in the referenced system, and its likelihood of being exploited. The following table summarizes the general expected classification according to impact and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Impact / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Security Review

Ojo Yield Risk Engine

Project Dashboard

Application Summary

Name	Yield Risk Engine
Repository	https://github.com/ojo-network/yield-risk-engine
Commit	0ac3939
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	29/05/2025
№ of Auditors	2
Review Time	0.4 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	1	1	0
None	3	3	0

Category Breakdown

Suggestion	3
Documentation	0
Bug	1
Optimization	0
Good Code Practices	0

Risk Section

Security Review

Ojo Yield Risk Engine

Risk Section

No risk has been identified.

Findings

Security Review

Ojo Yield Risk Engine

Findings

3S-Ojo-L01

OjoYieldRiskEngineAddresses mapping overwrites previous engine records when users create multiple engines

Id	3S-Ojo-L01
Classification	Low
Impact	Low
Likelihood	Medium
Category	Bug
Status	Addressed in #31a0856.

Description

The **OjoYieldRiskEngineFactory** contract maintains a mapping called **OjoYieldRiskEngineAddresses** that associates each user address with their created yield risk engine. This mapping is updated in the **createOjoYieldRiskEngine** function:

```
OjoYieldRiskEngineAddresses[msg.sender] = ojoYieldRiskEngine;
```

As implemented, this mapping only records the most recent engine created by each user, overwriting previous entries. When a user creates multiple engines over time, only the latest one remains accessible through this mapping. While the contract does emit an **OjoYieldRiskEngineCreated** event for each creation, directly querying a user's historical engines requires event log analysis rather than simple contract state reads.

Recommendation

Consider replacing the current one-to-one mapping with an array mapping that preserves all created engines.

3S-Ojo-N01

Repeated code in data retrieval functions

Id	3S-Ojo-N01
Classification	None
Category	Suggestion
Status	Addressed in #31a0856.

Description

In the **OjoYieldRiskEngine** contract, the functions **OjoYieldRiskEngine::latestRoundData** and **OjoYieldRiskEngine::getRoundData** both contain duplicated logic for capping the **answer** value. This duplication leads to lower code quality and maintainability, as any future changes to the capping logic would need to be made in multiple places. The purpose of these functions is to retrieve the latest or specific round data from the **BasePriceFeed**, while ensuring that the **answer** does not exceed a predefined **cappedAnswer**.

Recommendation

To improve maintainability and reduce code duplication, extract the capping logic into a separate function, **OjoYieldRiskEngine::capAnswer**, and use this function in both **OjoYieldRiskEngine::latestRoundData** and **OjoYieldRiskEngine::getRoundData**.

3S-Ojo-N02

Missing validation of answer value in **OjoYieldRiskEngine::initialize**

Id	3S-Ojo-N02
Classification	None
Category	Suggestion
Status	Addressed in #31a0856.

Description

The **OjoYieldRiskEngine::initialize** function is responsible for setting up the initial state of the **OjoYieldRiskEngine** contract, including setting the base price feed and yield cap. It also calculates the **cappedAnswer** based on the latest data from the base price feed. However, the function does not validate that the **answer** value returned by **BasePriceFeed.latestRoundData()** is greater than zero. This omission can lead to an incorrect calculation of **cappedAnswer** if the base price feed is malfunctioning or returning invalid data, potentially affecting the contract's functionality and the accuracy of the yield risk assessment.

Recommendation

To ensure the integrity of the **cappedAnswer** calculation, add a validation check to confirm that the **answer** value is greater than zero.

3S-Ojo-N03

Non-compliance with checks-effects-interactions pattern in refund logic

Id	3S-Ojo-N03
Classification	None
Category	Suggestion
Status	Addressed in #31a0856.

Description

The **OjoYieldRiskEngineFactory::createOjoYieldRiskEngine** function is responsible for creating a new instance of the **OjoYieldRiskEngine** contract. It requires the caller to have accepted terms and, if no free deployments are remaining, to pay a creation fee. The function attempts to refund any excess ETH sent by the caller after deducting the creation fee. However, the refund logic does not adhere to the checks-effects-interactions (CEI) pattern, as the external call to refund the excess ETH is made before the function's state changes are finalized. While there is no direct security issue identified, this non-compliance with the CEI pattern is a bad development practice.

Recommendation

To adhere to the checks-effects-interactions pattern and minimize potential reentrancy risks, the refund logic should be moved to the end of the function, after all state changes have been made, to ensure that the contract's state is updated before any external calls are made.