Three Sigma

Code Audit

Ojo A cross-chain oracle built as a blockchain

Disclaimer

Code Audit

Ojo A cross-chain oracle built as a blockchain

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 A cross-chain oracle built as a blockchain

Table of Contents

Summary

Code Audit

Ojo A cross-chain oracle built as a blockchain

Summary

Three Sigma audited Ojo in a 2 day engagement. The audit was conducted from 16-10-2024 to 17-10-2024.

Protocol Description

Ojo has created an exchange-rate oracle contract for LRTs curated by Mellow. This oracle is intended to output the exchange rate between an LRT and a particular base asset, while having an interface that is compatible with Morpho markets. Since Mellow accepts multiple asset types in its vaults, this contract takes the underlying TVL of the vault into consideration and is compatible with vaults which contain multiple currencies, rather than strictly pegging the LRT to the underlying token with a 1:1 ratio.

Scope

Code Audit

Ojo A cross-chain oracle built as a blockchain

Scope

CloneFactory

MellowPriceFeed

Assumptions

External libraries are assumed to be secure.

Methodology

Code Audit

Ojo A cross-chain oracle built as a blockchain

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 A cross-chain oracle built as a blockchain

Project Dashboard

Application Summary

Name	Ojo
Commit	e2ff96624afea24fbc9d79f5cc30f4ae2e8c530b
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	16-10-2024 to 17-10-2024
№ of Auditors	1
Review Time	2 days

Vulnerability Summary

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

None	1	1	0

Category Breakdown

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

Code Maturity Evaluation

Code Audit

Ojo A cross-chain oracle built as a blockchain

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 implementing safe arithmetic.
Centralization	Satisfactory. The protocol has no centralization points.
Code Stability	Satisfactory. The codebase was stable during the course of the audit.
Upgradeability	Weak. The price feed is not upgradeable.
Function Composition	Satisfactory. Functionality is well split into different helpers.
Front-Running	Satisfactory. No front-running issues were found.
Monitoring	Satisfactory. Events were correctly emitted.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Satisfactory. Although tests were not present the codebase is very simple.

Findings

Code Audit

Ojo A cross-chain oracle built as a blockchain

Findings

3S-Ojo-L01

Hardcoded **1e18** ratio calculation regardless of decimals

Id	3S-Ojo-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #ad88fe1.

Description

MellowPriceFeed gets the ratio by doing **answer = int256(uint256(ratiosX96[0])) * 1e18 / int256(managedRatiosOracle.Q96());**, hardcoding the decimals to **1e18**. However, the decimals are set in the **constructor** and stored in **priceFeedDecimals**, and could differ from **1e18**.

Recommendation

```
answer = int256(uint256(ratiosX96[0])) * 10**priceFeedDecimals /
int256(managedRatiosOracle.Q96());
```

3S-Ojo-N01

The number of underlying tokens may be modified in the **`Vault`** leading to inconsistent exchange rate

Id	3S-Ojo-N01
Classification	None
Category	Suggestion
Status	Addressed in 065a63f.

Description

The price is calculated based on the ratio from **`managedRatiosOracle.getTargetRatiosX96()`**. Currently the vault has only 1 underlying token, which means the ratio will always be **`1e18`**. However, the vault may add more tokens, which would change this.

Recommendation

Clarify the behaviour if more tokens are added.