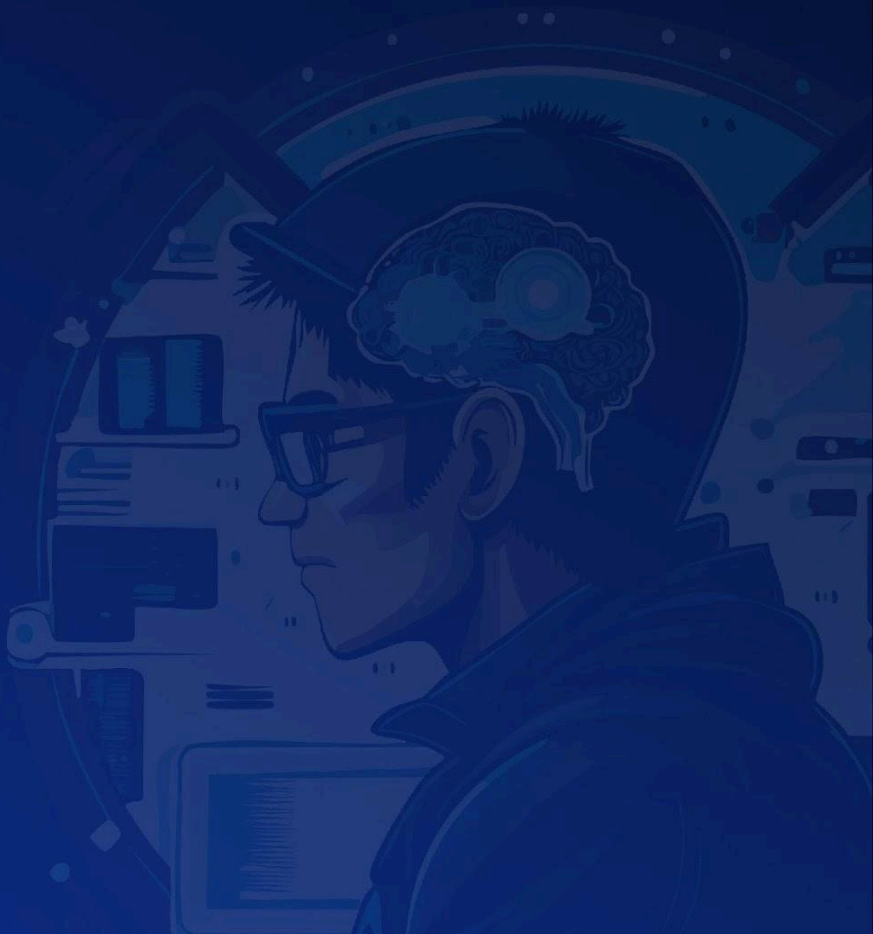
Mangrove Chainlink Aggregator

Security Review

Disclaimer

Security Review

Mangrove Chainlink Aggregator



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

Mangrove Chainlink Aggregator

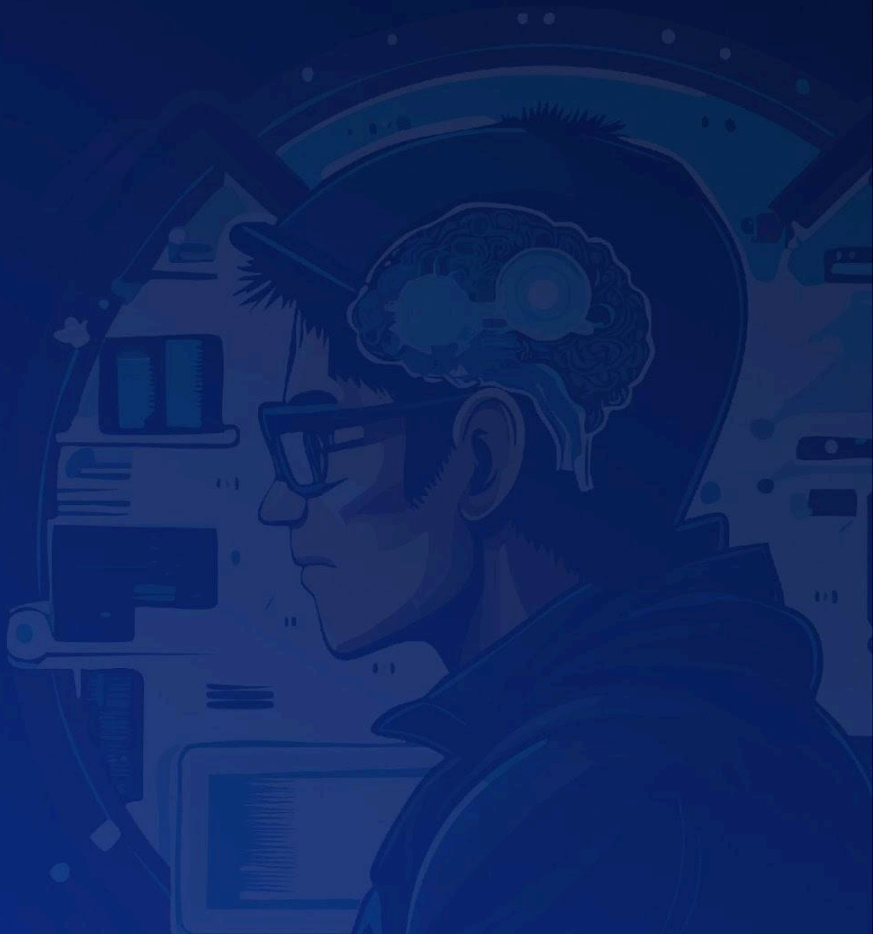
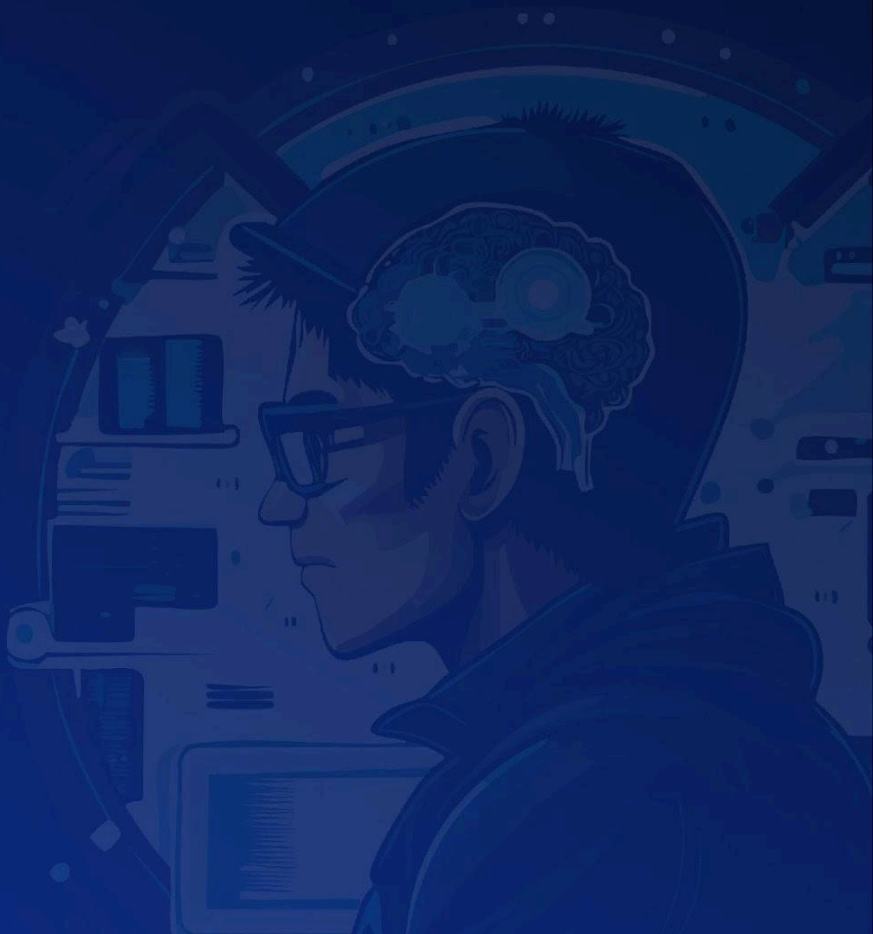


Table of Contents

Summary

Security Review

Mangrove Chainlink Aggregator



Summary

Three Sigma audited Mangrove in a 0.6 person week engagement. The audit was conducted on 11/11/2025.

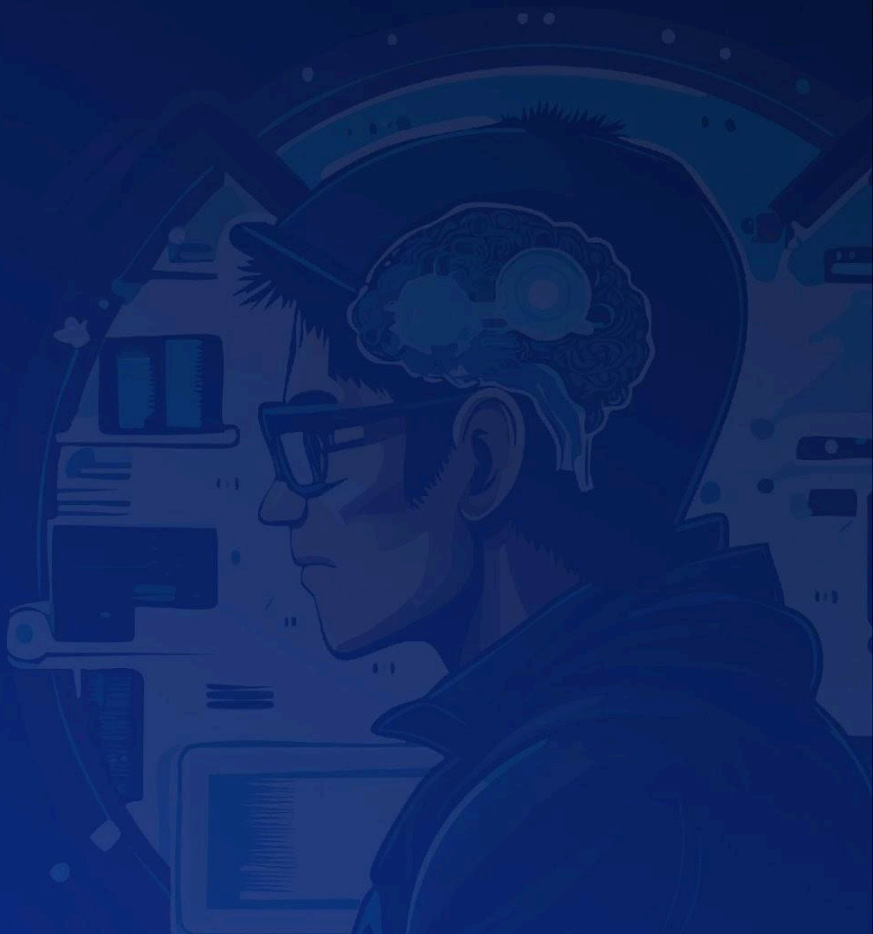
Protocol Description

Mangrove is building the next generation of trading infrastructure, featuring a fully on-chain, composable CLOB DEX designed for high efficiency and seamless integration across the ecosystem.

Scope

Security Review

Mangrove Chainlink Aggregator



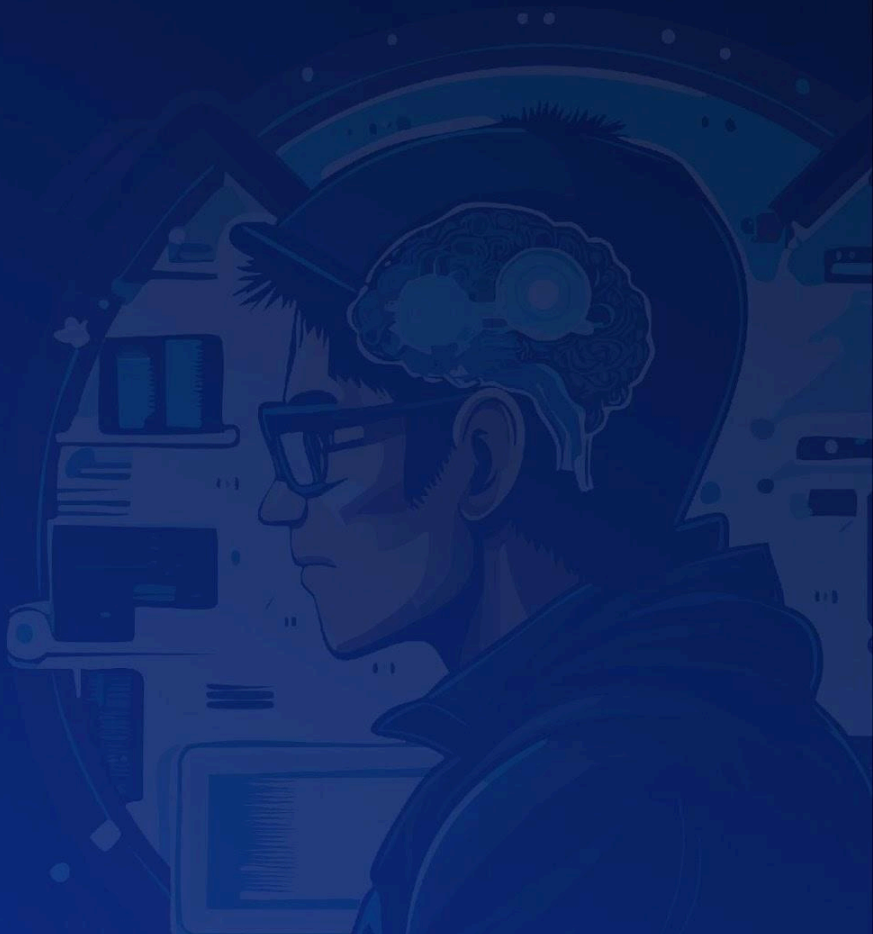
Scope

Filepath	nSLOC
src/VaultsV2Feed.sol	83
src/libraries/V2VaultPricing.sol	39
src/libraries/V2VaultReader.sol	15
src/libraries/InitialParameters.sol	26
src/libraries/ChainlinkConsumer.sol	13
Total	**176**

Methodology

Security Review

Mangrove Chainlink Aggregator



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

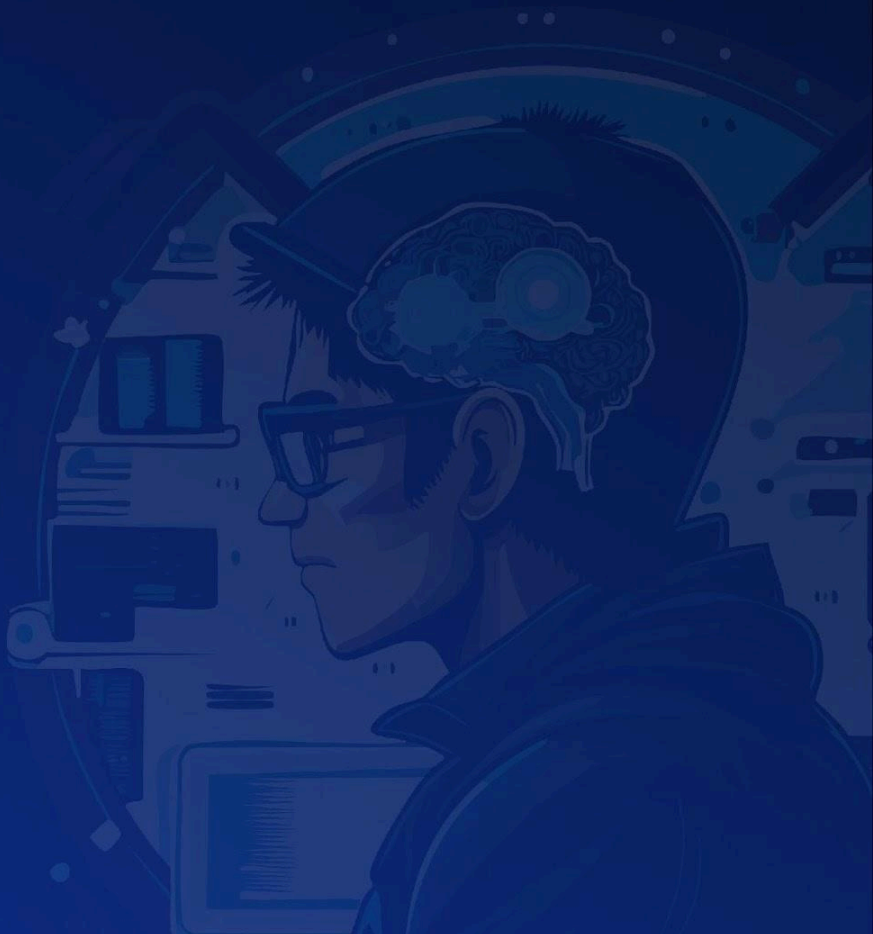
In this audit, we classify findings based on Immunefi's Vulnerability Severity Classification System (v2.3) as a guideline. The final classification considers both the potential impact of an issue, as defined in the referenced system, and its likelihood of being exploited. The following table summarizes 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

Mangrove Chainlink Aggregator



Project Dashboard

Application Summary

Name	Mangrove
Repository	https://github.com/mangrovedao/vaults-v2-chainlink-adapter
Commit	259f3b6
Language	Solidity
Platform	Base, Arbitrum

Engagement Summary

Timeline	11/11/2025
№ of Auditors	3
Review Time	0.6 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	1	0	1

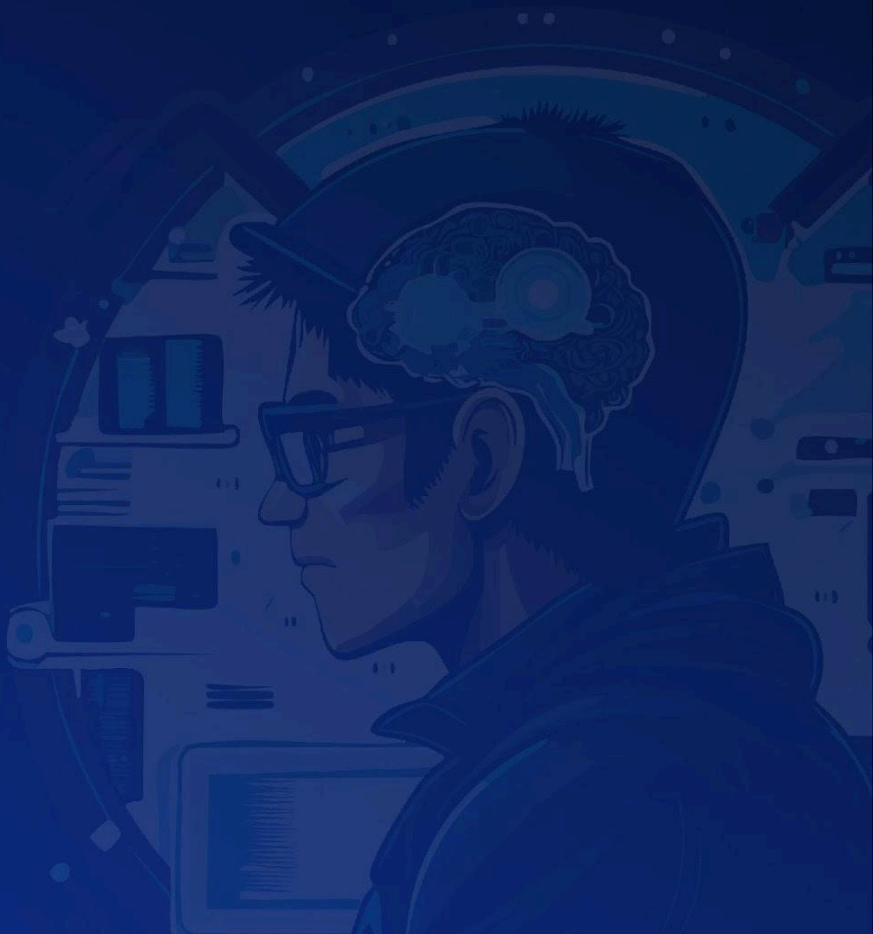
Category Breakdown

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

Risk Section

Security Review

Mangrove Chainlink Aggregator



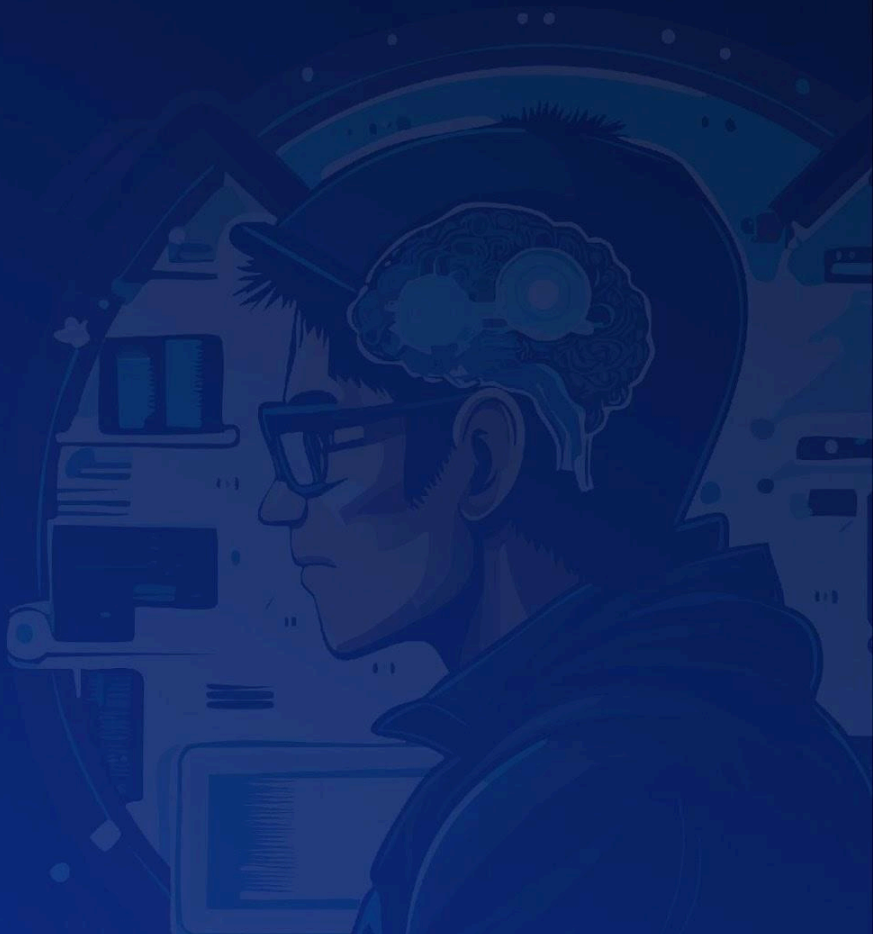
Risk Section

No risks were identified.

Findings

Security Review

Mangrove Chainlink Aggregator



Findings

3S-Mangrove-L01

Invalid timestamp returned when vault has zero total supply

Id	3S-Mangrove-L01
Classification	Low
Impact	Low
Likelihood	Low
Category	Bug
Status	Addressed in #8143edb.

Description

The **V2VaultPricing::price** function computes the price per vault share by combining Chainlink price feed data with on-chain vault balances. This function is called by **VaultsV2Feed::latestRoundData** and other view functions to provide Chainlink-compatible price feed data for vault shares.

When the vault's **totalSupply** is zero (no shares have been issued), the function returns a valid price calculation based on the quote asset price. However, it incorrectly returns **updatedAt** as **0** instead of propagating the actual timestamp from the underlying Chainlink price feeds. This zero timestamp can be misinterpreted by feed consumers as stale or invalid data, even though the price calculation itself is valid and based on fresh feed data.

The underlying price feeds (**baseFeed** and **quoteFeed**) are queried via the **answers** function, which retrieves legitimate timestamps indicating when those feeds were last updated. These timestamps represent the freshness of the price data and should be returned to consumers regardless of whether the vault has issued shares.

Recommendation

Return the actual **updatedAt** timestamp from the underlying price feeds instead of hardcoding it to **0**. Since the price calculation when **totalSupply == 0** depends solely on the quote feed, returning the quote feed's timestamp would be most accurate. However, returning the combined timestamp (minimum of both feeds) already computed by the **answers** function is also acceptable and maintains consistency with the non-zero case.

```
function price(
  address vault,
  address baseFeed,
  address quoteFeed,
  uint256 baseMultiplier,
  uint256 quoteMultiplier,
  uint256 baseDivider,
  uint256 quoteDivider,
  uint256 sharesMultiplier
) internal view returns (int256 _price, uint256 updatedAt) {
  unchecked {
    uint256 totalSupply = vault.totalSupply();
    int256 baseAnswer;
    int256 quoteAnswer;
    (baseAnswer, quoteAnswer, updatedAt) = answers(baseFeed, quoteFeed);
    if (totalSupply == 0) {
      // casting to 'int256' is safe because quoteMultiplier is a small value
      // forge-lint: disable-next-line(unsafe-typecast)
-     return (quoteAnswer * int256(quoteMultiplier), 0);
+     return (quoteAnswer * int256(quoteMultiplier), updatedAt);
    }
    uint256 _value = value(vault, baseAnswer, quoteAnswer, baseMultiplier, quoteMultiplier,
baseDivider, quoteDivider);
    _price = int256(_value.mulDiv(sharesMultiplier, totalSupply));
  }
}
```

3S-Mangrove-N01

Missing L2 sequencer uptime validation in price feed queries

Id	3S-Mangrove-N01
Classification	None
Category	Suggestion
Status	Acknowledged with note

Description

The **VaultsV2Feed** contract implements a Chainlink-compatible price feed adapter that computes vault share prices by aggregating data from underlying Chainlink price feeds for base and quote assets. The contract exposes functions such as **latestRoundData()**, **latestAnswer()**, and **getRoundData()** that query current prices from Chainlink feeds and perform calculations to derive the vault share price.

When operating on Layer 2 networks such as Arbitrum or Base, the contract does not validate the sequencer's uptime status before returning price data. If the L2 sequencer experiences downtime and subsequently comes back online, stale prices may be returned during the grace period, potentially leading to incorrect valuations of vault shares. Chainlink provides sequencer uptime feeds specifically for L2 networks to address this concern.

Without this validation, consuming contracts that rely on **VaultsV2Feed** for pricing decisions (such as liquidations, swaps, or collateral valuations) may operate on outdated data immediately after sequencer recovery.

Reference: Chainlink L2 Sequencer Uptime Feeds

Recommendation

There are two approaches to mitigate this issue:

Option 1: Integrate sequencer uptime validation into the feed contract

Implement L2 sequencer uptime validation directly in **VaultsV2Feed** to ensure price data is only returned when the sequencer is operational and the grace period has elapsed.

Option 2: Document the behavior and delegate responsibility to consuming contracts

If the design philosophy prefers to keep **VaultsV2Feed** as a simple data provider without sequencer checks, document this requirement clearly so consuming contracts understand they must implement sequencer uptime validation.

Client note

We'll indeed leave the feed as is and let the consuming contracts deal with these kind of errors.

The goal of this chain link compatible price oracle is to provide a price feed as is a basic Chainlink oracle price feed. Should a parent feed go down or have a latency, it is expected that ours will as well. It is also expected that the consuming contracts should implement safety behaviors that will check for L2s downtime or even staleness of the data from the forwarded timestamp (this is why the forwarded timestamp is the oldest).