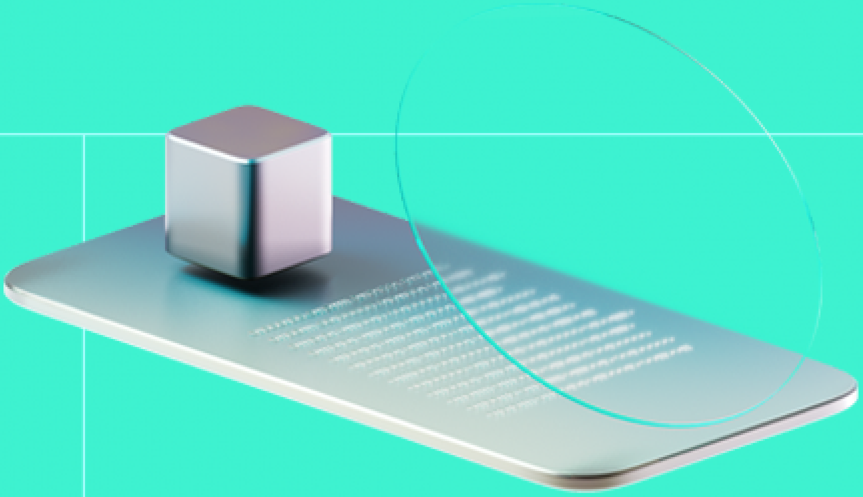
Smart Contract Code Review And Security Analysis Report



Customer: ChainSight

Date: 23/04/2025

We express our gratitude to the ChainSight team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Chainsight oracles are modular, on-chain data pipelines composed of indexers, calculators, and relayers that fetch, process, and deliver external data to smart contracts. They support reusable data components across chains .

Document

Name	Smart Contract Code Review and Security Analysis Report for ChainSight
Audited By	Lukasz Mikula
Approved By	Ataberk Yavuzer
Website	https://chainsight.network
Changelog	16/04/2025 - Preliminary Report
	23/04/2025 - Final Report
Platform	Ethereum
Language	Solidity
Tags	Oracle
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository #1	https://github.com/horizonx-tech/chainsight-management-oracle
Commit #1	955bad45318c98549525bbe70fa19ba7bbff893b
Remediation Commit #1	955bad45318c98549525bbe70fa19ba7bbff893b
Repository #2	https://github.com/horizonx-tech/chainsight-multisource-oracle
Commit #2	86aa2b1576c9e4f2c81ae5883f6c814a0c7d89ad

Review Scope

Remediation Commit #2	0e5cc1b5287e2fc1b041e0e0bbeeccd7edde9c38

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

5	5	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	1
Low	2

Vulnerability	Severity
F-2025-9838 - Fallback to newest stale price is possible	Medium
F-2025-9839 - Single source failure causes complete oracle failure	Low
F-2025-9841 - Hard dependency on Pyth for specific interface methods	Low
F-2025-9840 - Missing event emissions	Info
F-2025-9842 - No duplicate checks when adding ChainSight sources	Info

Documentation quality

- A detailed documentation about the protocol was provided

Code quality

- The code has multiple functions that are very similar to each other; they could be merged into one function.
- A redundant code was found in the project

Test coverage

- The tests are present, but were not working due to unresolved dependency issues

Table of Contents

System Overview

ChainSight Oracle System consists of two main contracts:

Oracle.sol — A data storage contract that allows addresses to record and retrieve timestamped data.

It has the following attributes:

- Built with OpenZeppelin's upgradeable proxy pattern
- Stores arbitrary data with associated timestamps
- Provides read functions for various data types (strings, uint256, int256)
- Each address can only write data for itself

MultiSourceOracle.sol — An aggregator that combines price data from multiple sources:

- Integrates Chainlink, Pyth, and ChainSight oracles
- Uses time-weighted averaging with configurable parameters
- Implements median-based outlier detection
- Normalizes prices to a consistent decimal format (default: 8 decimals)
- Includes fallback mechanisms for handling stale data

Privileged roles

- Owner can configure price sources (add/remove/modify)
- Owner can adjust system parameters (stale threshold, decay rate, etc.)
- Owner can toggle outlier detection
- Owner can pause and unpause the system

Potential Risks

- **Data Source Failures:** If any single oracle source reverts, the entire aggregation process will fail due to a lack of try/catch mechanism.
- Price Manipulation: Outlier detection is bypassed when fewer than 3 fresh data sources are available, creating vulnerability during periods of limited oracle availability.
- **Precision Loss:** Conversion between different decimal precisions uses truncation rather than rounding, creating systematic bias, particularly for non-USD-denominated pairs.
- **Silent Stale Data Fallback:** When all sources are stale, the system automatically falls back to the newest stale price without notifying consumers of this transition.
- **Time-Weight Exploitation:** The exponential decay weighting system gives significantly higher influence to recent timestamps, potentially allowing manipulation by controlling the most recently updated source.

Findings

Vulnerability Details

F-2025-9838 - Fallback to newest stale price is possible - Medium

Description:

When all sources are stale, the contract uses the newest stale price as a fallback mechanism in the `_fallbackNewest()` function. This is dangerous, because if a condition where all prices are stale occurs, that means something bad had happened to the protocol - as such situation is unlikely.
Hence, in such situation, no stale price should be used as there is a high risk it will be incorrect.

```
  // 3) If none fresh => fallback newest stale
if (freshCount == 0) {
    return _fallbackNewest(list);
}
```

Assets:

- src/MultiSourceOracle.sol [https://github.com/horizonx-tech/chainsight-multisource-oracle]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Semi-Dependent

Complexity: Complex

Severity: Medium

Recommendations

Remediation: Rather than defaulting to the newest stale price, the contract should revert when all sources are stale, forcing dependent protocols to handle the unavailability of fresh price data explicitly.

Resolution: Fixed in commit `1078044`. The contract now controls fallback to latest stale price with a new variable `allowStaleFallback`. If toggled off, all stale sources cause a revert.

F-2025-9839 - Single source failure causes complete oracle failure - Low

Description:

In `_collectAllSources()`, any single source that reverts (Chainlink, Pyth, or any ChainSight oracle) will cause the entire aggregation to fail, as there is no try/catch mechanism to handle individual source failures.

This creates a significant reliability risk as the oracle becomes only as reliable as its least reliable data source, undermining the redundancy benefit of having multiple sources. This may happen, for example, if just one of the sources is paused.

```
[…]
         // ChainSight
        for (uint256 i = 0; i < chainsightSources.length; i++) {
            (uint256 csPrice, uint64 csTime) = chainsightSources[i].oracle.readAsUint256WithTimestamp(
                chainsightSources[i].sender, chainsightSources[i].key
            );
            // cPrice is unsigned => no negative check
            uint256 csScaled = _scaleChainSightPrice(csPrice, chainsightSources[i].decimals);
            uint256 csWeight = _validWeight(csTime);
            list[idx] = SourceData(csScaled, csWeight, csTime);
            idx++;
        }
[…]
```

Assets:

- src/MultiSourceOracle.sol [https://github.com/horizonx-tech/chainsight-multisource-oracle]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Semi-Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Implement try/catch blocks around each external call to handle individual source failures gracefully, allowing the aggregator to continue operating with the remaining valid sources.

Resolution: The try/catch blocks are implemented now. Valid sources are tracked. **Fixed** in commit `819cc0c`.

F-2025-9841 - Hard dependency on Pyth for specific interface methods - Low

Description: Methods like `getPrice()` and `getPriceUnsafe()` require Pyth to be configured (`address(pyth) != address(0)`) to function, making these interfaces unusable even if other price sources are available.

If Pyth becomes unavailable or is intentionally disabled, protocols specifically relying on these interface methods will fail even though the oracle could still aggregate prices from other sources.

Assets:

- src/MultiSourceOracle.sol [https://github.com/horizonx-tech/chainsight-multisource-oracle]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Modify these methods to work with aggregated price data regardless of whether Pyth is configured, maintaining interface compatibility.

Resolution: The requirement has been removed in commit `098bcb9`.

F-2025-9840 - Missing event emissions - Info

Description: The contract lacks event emissions for critical parameter updates including `setChainlinkFeed()`, `setPythFeed()`, `addChainSightSource()`, `clearAllChainSightSources()`, `setAggregatorDecimals()`, `setStaleThreshold()`, `setLambda()`, `setMaxPriceDeviationBps()`, and `setOutlierDetectionEnabled()`.

The lack of events makes it difficult to monitor and track important contract configuration changes off-chain, reducing transparency.

Assets:

- src/MultiSourceOracle.sol [https://github.com/horizonx-tech/chainsight-multisource-oracle]

Status: Fixed

Classification

Impact: 1/5

Likelihood: 1/5

Exploitability: Independent

Complexity: Simple

Severity: Info

Recommendations

Remediation: Add appropriate events for all state-changing operations to ensure proper off-chain monitoring capabilities.

Resolution: Emission of events was added in commit `907f733`.

F-2025-9842 - No duplicate checks when adding ChainSight sources - Info

Description: The constructor and `addChainSightSource()` function does not check for duplicate ChainSight oracles, which could lead to the same source being added multiple times.

Duplicate sources would be given more weight in the aggregation than intended, potentially skewing results toward that particular data source.

Status: Fixed

Classification

Impact:	3/5
Likelihood:	1/5
Exploitability:	Dependent
Complexity:	Complex
Severity:	Info

Recommendations

Remediation: Add checks to prevent duplicate ChainSight sources from being added to the contract.

Resolution: The duplicates are now being checked. **Fixed** in commit `d97d4c2`.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository #1	https://github.com/horizonx-tech/chainsight-management-oracle
Commit #1	955bad45318c98549525bbe70fa19ba7bbff893b
Repository #2	https://github.com/horizonx-tech/chainsight-multisource-oracle
Commit #2	86aa2b1576c9e4f2c81ae5883f6c814a0c7d89ad
Whitepaper	N/A
Requirements	N/A
Technical Requirements	N/A

Asset	Type
contracts/Oracle.sol [https://github.com/horizonx-tech/chainsight-management-oracle]	Smart Contract
src/MultiSourceOracle.sol [https://github.com/horizonx-tech/chainsight-multisource-oracle]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.