Quantstamp

API3 - Data Feed Proxy Combinators

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Oracle Wrapper
Timeline	2025-06-09 through 2025-06-13
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README
Source Code	• https://github.com/api3dao/data-feed-proxy-combinators #3503ca8
Auditors	• Adrian Koegl Auditing Engineer • Hytham Farah Auditing Engineer • Yamen Merhi Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	4 Acknowledged: 4
High severity findings	0
Medium severity findings	2 Acknowledged: 2
Low severity findings	1 Acknowledged: 1
Undetermined severity findings	0
Informational findings	1 Acknowledged: 1

Summary of Findings

Quantstamp has audited API3's Data Feed Proxy Combinator codebase, which offers different ways to adapt oracle price data feeds to other formats. Each contract in scope wraps either a Chainlink or API3 data feed and exposes both of their interfaces to transform the oracle data in the following ways:

- `InverseApi3ReaderProxyV1` inverts the oracle price.
- `ScaledApi3FeedProxyV1` exposes a Chainlink interface for API3 data feeds.
- `NormalizedApi3ReaderProxyV1` exposes an API3 interface for Chainlink data feeds.
- `ProductApi3ReaderProxyV1` returns the product of two oracles.
- `PriceCappedApi3ReaderProxyV1` caps the price with a lower and/or upper bound.

The security of the oracle contracts mostly depends on the context of their integration and usage. However, we have outlined two medium severity issues that may introduce security concerns for integrating protocols.

Overall, the contracts are well-written and security best practices are followed.

Fix Review Update: During the fix review, the client has acknowledged all four findings and elaborated on them in the NatSpec documentation. The two suggestions were successfully fixed.

ID	DESCRIPTION	SEVERITY	STATUS
DFP-1	Silent Truncation on `int224` Cast without Bounds Checks	Medium	Acknowledged
DFP-2	Aggregation of Stale Price Feed(s)	Medium	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
DFP-3	Silent Underflow to Zero	Low	Acknowledged
DFP-4	Precision Loss on Value Downscaling	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

In scope for this audit are the four outlined oracle wrappers and their interfaces.

Files Included

Repo: https://github.com/api3dao/data-feed-proxy-combinators

Files:

- contracts/InverseApi3ReaderProxyV1.sol
- contracts/NormalizedApi3ReaderProxyV1.sol
- contracts/ProductApi3ReaderProxyV1.sol
- contracts/PriceCappedApi3ReaderProxyV1.sol

- contracts/adapters/ScaledApi3FeedProxyV1.sol

Files Excluded

Repo: https://github.com/api3dao/data-feed-proxy-combinators

Files:

- contracts/vendor
- contracts/test

Operational Considerations

- The deployer of the wrapper contracts are responsible for the correct configuration.
- The wrapper contracts utilize u32 for timestamps and are, therefore, not usable after 2106.
- Some of the wrapper contracts may revert due to integer overflow (especially `ProductApi3ReaderProxyV1.sol`) for high prices.
- These combinator proxies are designed as lightweight wrappers. They do not perform additional validation on the data (e.g., value sanity checks or timestamp verification) returned by the underlying dAPIs or external feeds. The responsibility for data quality and timeliness rests with the source oracle or feed provider.
- All proxy contracts assume the upstream `IApi3ReaderProxy` feed returns a valid and fresh value conforming to 18-decimal fixed-point format. Malformed or stale upstream values may silently propagate and result in incorrect downstream outputs.
- `NormalizedApi3ReaderProxyV1` relies on static metadata (`decimals()`) from Chainlink feeds. It computes a normalization factor once during deployment and assumes feed decimals remain stable throughout the contract's lifetime. Feed upgrades that change decimal precision would require redeployment of this proxy.
- The `PriceCappedApi3ReaderProxyV1` contract permits setting identical or nearly identical upper and lower bounds (e.g., upperBound == lowerBound), which effectively creates a static price feed regardless of the underlying proxy's value fluctuations. While this functionality is intentional as mentioned in the comments ("To configure a fixed price, set lowerBound_ and upperBound_ to the same desired price"), users should be explicitly informed about this capability and its implications for price feed behavior in documentation, especially when minimal price variation could impact dependent protocols.

Key Actors And Their Capabilities

No privileged roles are defined or used and all contracts are immutable.

Findings

DFP-1
Silent Truncation on `int224` Cast without Bounds Checks

Medium | Acknowledged

Update

Acknowledged in: `94ae5e9`.
The client provided the following explanation:

```
The absence of explicit bounds checks for the int256 to int224 cast in
ProductApi3ReaderProxyV1.sol and NormalizedApi3ReaderProxyV1.sol is a deliberate trade-off for
gas optimization in the read() functions. This design choice means that silent truncation may
occur if the intermediate int256 result exceeds int224 limits, a behavior now clearly
documented in their respective NatSpec comments.
  For InverseApi3ReaderProxyV1.sol, the read() function's formula was updated to value =
int224(1e36) / baseValue;. With this change, the division occurs directly between int224
types, and thus the specific concern regarding silent truncation from an intermediate int256
cast is not directly applicable to this revised calculation. Its NatSpec comment has been
updated to accurately reflect the current operation.
```

File(s) affected: `ProductApi3ReaderProxyV1.sol`, `NormalizedApi3ReaderProxyV1.sol`, `InverseApi3ReaderProxyV1.sol`

Description: Multiple contracts perform unchecked casts from `int256` to `int224`, which can silently truncate values exceeding the `int224` range, leading to incorrect and misleading outputs. Solidity does not revert on narrowing conversions, so these casts may discard high-order bits without warning:

- `ProductApi3ReaderProxyV1.read()` : Multiplies two proxy values and divides by 1e18 before casting to `int224` without checking bounds, risking overflow and silent truncation.
- `NormalizedApi3ReaderProxyV1.read()` : Scales a Chainlink feed value (up or down) and casts to `int224` without verifying that the result fits within range.
- `InverseApi3ReaderProxyV1.read()` : Inverts a proxy value using 1e36 divided by the feed value, then casts to `int224`, with a misleading comment implying a revert on overflow.

Recommendation: Insert explicit range checks before casting to `int224` to prevent silent truncation, or use `SafeCast.toInt224()` from OpenZeppelin. Update comments to correctly describe casting behavior and eliminate references to nonexistent overflow reverts.

DFP-2 Aggregation of Stale Price Feed(s)

Medium Acknowledged

Update

Acknowledged in: `e0595d4` .
The client provided the following explanation:

```
The ProductApi3ReaderProxyV1.read() function intentionally returns block.timestamp. This
design choice by Api3 reflects that the timestamp accurately marks the on-chain computation
time of the derived product value. Aggregating timestamps from underlying feeds is avoided due
to complexity and the potential for misinterpretation. Integrators are responsible for
checking the staleness of individual underlying feeds directly if required, as per Api3's
integration guidelines (https://docs.api3.org/dapps/integration/contract-
integration.html#using-timestamp). The contract's NatSpec documents this behavior.
```

File(s) affected: `ProductApi3ReaderProxyV1.sol`

Description: The `ProductApi3ReaderProxyV1` returns the product of possibly stale data feed sources. It returns `block.timestamp` as the timestamp for the new value, masking the age of the underlying data feeds. A consuming protocol utilizing this wrapper cannot apply their freshness policy to the returned price, leading to outdated prices and creating an opportunity for attackers. This could lead to protocols unintentionally accepting stale price data and attackers exploiting those protocols.

Recommendation: Protocols integrating the `ProductApi3ReaderProxyV1` may want to apply their freshness policy to the least recent price. Therefore, consider propagating the smaller timestamp of the underlying feeds (`MIN(timestamp1, timestamp2)`).

DFP-3 Silent Underflow to Zero

Low Acknowledged

Update

Acknowledged in: `2ff87d9` .
The client provided the following explanation:

```
The potential for the read() function in InverseApi3ReaderProxyV1.sol to return 0 due to
integer division underflow is an intentional design choice. This prioritizes maximum gas
efficiency for this "highly unlikely" scenario, rather than adding an explicit check to
revert. This behavior is now documented in the contract's NatSpec comment for the read()
function.
```

File(s) affected: `InverseApi3ReaderProxyV1.sol`

Description: The current implementation of price inversion carries the marginal risk of underflowing to zero. In case the data feed value becomes larger than `1e36` , the inverted price will incorrectly return zero, enabling various exploits for protocol utilizing the price inversion mechanism. While this is highly unlikely for data feeds like ETH/USD (ETH would have to be worth one quintillon USD), it may occur for data feeds that, e.g., involve hyper-inflated asset or exotic asset pairs.

Recommendation: Revert the `read()` function whenever the inverted value is equal to zero. Consider introducing a `UnderflowToZero` error and throwing it in that case.

DFP-4 Precision Loss on Value Downscaling

Informational Acknowledged

Update

Acknowledged in: `e70c71f` .
The client provided the following explanation:

```
The potential for precision loss when downscaling values in ScaledApi3FeedProxyV1.sol is an intentional design choice. This approach prioritizes flexibility for integrators who may require feeds with fewer decimals, rather than restricting the contract's scaling capabilities. The responsibility for assessing and managing the impact of any such precision loss rests with the integrating protocol. This behavior, including the use of integer division which truncates during downscaling, is now clearly documented in the contract's main NatSpec.
```

File(s) affected: `ScaledApi3FeedProxyV1.sol`

Description: The `ScaledApi3FeedProxyV1` wrapper allows to downscale any data feed to a lower number of decimals. This loss of precision may result in inaccurate pricing, which can potentially be exploited by an attacker. While it is the responsibility of the integrating protocol, downscaling precision may be a risk in general.

Recommendation: Consider restricting the price feed to only permit upscaling, disallowing any downscaling.

Auditor Suggestions

S1 Unlocked Pragma

Fixed

Update

Addressed in: `ecfd948` . All pragma's were fixed to `0.8.27` .

File(s) affected: `NormalizedApi3ReaderProxyV1.sol` , `InverseApi3ReaderProxyV1.sol` , `ScaledApi3ReaderProxyV1.sol` , `ProductApi3ReaderProxyV1.sol` , `PriceCappedApi3ReaderProxyV1.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity ^0.8.27` . The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

S2 Error Handling

Fixed

Update

Addressed in: `16751b3` .
The client provided the following explanation:

```
Reverting with the custom DivisionByZero() error is more gas-efficient and provides clearer error semantics than Solidity's default panic, with minimal gas impact on successful reads.
```

File(s) affected: `InverseApi3ReaderProxyV1.sol` , `IInverseApi3ReaderProxyV1.sol`

Description: If the price of the underlying data feed is zero, the `read()` function will revert with an unhandled error. The `IInverseApi3ReaderProxyV1` declares the `DivisionByZero()` error, however, it is not currently used.

Recommendation: We recommend explicitly throwing the `DivisionByZero()` error when the data feed value is zero.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Test Suite Results

All 71 test cases were successfully run.

```
InverseApi3ReaderProxyV1
  constructor
    proxy is not zero address
      ✔ constructs (165ms)
    proxy is zero address
      ✔ reverts
  read
    ✔ reads the inverse rate
  latestAnswer
    ✔ returns proxy value
  latestTimestamp
    ✔ returns proxy value
  latestRound
    ✔ reverts
  getAnswer
    ✔ reverts
  getTimestamp
    ✔ reverts
  decimals
    ✔ returns 18
  description
    ✔ returns empty string
  version
    ✔ returns 4915
  getRoundData
    ✔ reverts
  latestRoundData
    ✔ returns approximated round data

NormalizedApi3ReaderProxyV1
  constructor
    feed is not zero address
      feed does not have 18 decimals
        ✔ constructs (58ms)
      feed has 18 decimals
        ✔ reverts
    feed is zero address
```

```
      ✔ reverts
    read
      ✔ reads the normalized to 18 decimals rate (46ms)
    latestAnswer
      ✔ returns proxy value
    latestTimestamp
      ✔ returns proxy value
    latestRound
      ✔ reverts
    getAnswer
      ✔ reverts
    getTimestamp
      ✔ reverts
    decimals
      ✔ returns 18
    description
      ✔ returns empty string
    version
      ✔ returns 4916
    getRoundData
      ✔ reverts
    latestRoundData
      ✔ returns approximated round data

  PriceCappedApi3ReaderProxyV1
    constructor
      proxy is not zero address
        lowerBound is not negative
          upperBound is greater or  equal to lowerBound
            ✔ constructs (164ms)
          upperBound is less than lowerBound
            ✔ reverts
        lowerBound is negative
          ✔ reverts
      proxy is zero address
        ✔ reverts
    read
      ✔ reads the capped rate (66ms)
    latestAnswer
      ✔ returns proxy value
    latestTimestamp
      ✔ returns proxy value
    latestRound
      ✔ reverts
    getAnswer
      ✔ reverts
    getTimestamp
      ✔ reverts
    decimals
      ✔ returns 18
    description
      ✔ returns empty string
    version
      ✔ returns 4915
    getRoundData
      ✔ reverts
    latestRoundData
      ✔ returns approximated round data

  ProductApi3ReaderProxyV1
```

```
    constructor
      proxy1 is not zero address
        proxy2 is not zero address
          proxy1 is not the same as proxy2
            ✔ constructs (190ms)
          proxy1 is the same as proxy2
            ✔ reverts
        proxy2 is zero address
          ✔ reverts
      proxy1 is zero address
        ✔ reverts
    read
      ✔ reads the product of the proxy rates (43ms)
    latestAnswer
      ✔ returns proxy value (41ms)
    latestTimestamp
      ✔ returns proxy value (40ms)
    latestRound
      ✔ reverts
    getAnswer
      ✔ reverts
    getTimestamp
      ✔ reverts
    decimals
      ✔ returns 18
    description
      ✔ returns empty string
    version
      ✔ returns 4914
    getRoundData
      ✔ reverts
    latestRoundData
      ✔ returns approximated round data (41ms)

  ScaledApi3FeedProxyV1
    constructor
      proxy is not zero address
        targetDecimals is not invalid
          targetDecimals is not 18
            ✔ constructs (137ms)
          targetDecimals is 18
            ✔ reverts
        targetDecimals is invalid
          ✔ reverts
      proxy is zero address
        ✔ reverts
    latestAnswer
      ✔ returns proxy value
    latestTimestamp
      ✔ returns proxy value
    latestRound
      ✔ reverts
    getAnswer
      ✔ reverts
    getTimestamp
      ✔ reverts
    decimals
      ✔ returns 18
    description
      ✔ returns empty string
```

```
    version
      ✔ returns 4917
    getRoundData
      ✔ reverts
    latestRoundData
      ✔ returns approximated round data


  71 passing (2s)
```

Code Coverage

The 71 test cases achieve an average of 95% statement coverage and 84% branch coverage. We recommend increasing the statement coverage of `PriceCappedApi3ReaderProxyV1` to 100%. Otherwise, while a branch coverage of 100% is recommended, good testing practices are followed.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	93.75	84.38	97.96	96.84	
InverseApi3ReaderProxyV1.sol	100	100	100	100	
NormalizedApi3ReaderProxyV1.sol	100	75	100	96.15	33
PriceCappedApi3ReaderProxyV1.sol	81.82	83.33	92.31	93.1	85,86
ProductApi3ReaderProxyV1.sol	100	100	100	100	
contracts/adapters/	100	83.33	100	100	
ScaledApi3FeedProxyV1.sol	100	83.33	100	100	
contracts/adapters/interfaces/	100	100	100	100	
IScaledApi3FeedProxyV1.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IInverseApi3ReaderProxyV1.sol	100	100	100	100	
INormalizedApi3ReaderProxyV1.sol	100	100	100	100	
IPriceCappedApi3ReaderProxyV1.sol	100	100	100	100	
IProductApi3ReaderProxyV1.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	95	84.09	98.36	97.52	

Changelog

- 2025-06-13 - Initial report
- 2025-06-24 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.