Enshrined Oracle Secure Code Review

Technical Report

Thorchain

02 September 2025
Version: 1.1

Kudelski Security – Nagravision Sàrl

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

For Public Release

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Thorchain (“the Client”) engaged Kudelski Security (“Kudelski”, “We”) to perform the Enshrined Oracle Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

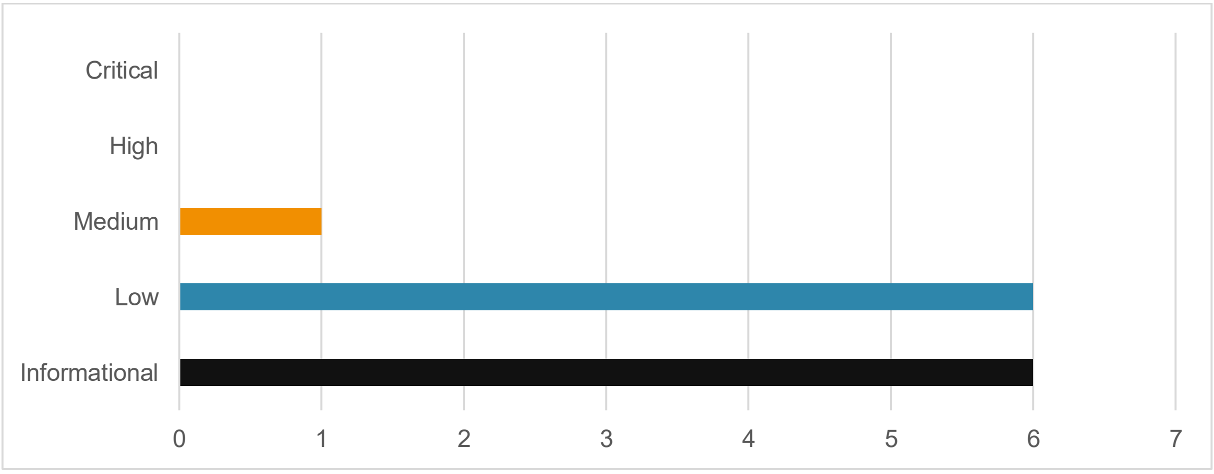
The review took place between 14 August 2025 and 29 August 2025, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

Key Findings

The following are the major themes and issues identified during the audit period. These, along with other items within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Unbuffered Channel Deadlock
- Empty Result Not Checked
- Check Ticker Never Stopped



Findings ranked by severity

1. PROJECT SUMMARY

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

1.1 Context

The `Enshrined oracle` implements an integrated price oracle, using the p2p gossip protocol of enshrined bifrost. The oracle has a multitude of price providers, separate processes that connect to a specific CEX and polls or subscribes to price feeds, which are then combined to a resulting USD price for each asset.

1.2 Scope

The scope consisted in specific Golang files and folders located at:

- Merge request: https://gitlab.com/thorchain/thornode/-/merge_requests/4144
- Final commit: fcc2ad320d2e1d305b6794812562accb34393b81

After the initial report (V1.0) was delivered, the Client addressed all vulnerabilities in the following codebase revision:

- Final commit: f1e15355cf31026b3e2a0ed26d02b6e963703d9f

The goal of the evaluation was to perform a security audit on the source code.

- No additional systems or resources were in scope for this assessment.
- The dependencies are out of scope of the review.
- Test codes are out of scope.

1.3 Remarks

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The code is well structured in general.
- Quick and open communication via Telegram
- The developers have made a careful and in-depth analysis of their project.
- We had regular and enriching technical exchanges on various topics.

1.4 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in Chapter Vulnerability Scoring System of this document.

2. STATIC CODE ANALYSIS

2.1 Semgrep

`Semgrep` (v1.99.0) identified 3 warnings from the code in scope. They are listed as below. The first warning is false positive, as commented by the Client. It is protected by existing `io.LimitReader` protection mechanism.

TITLE	DESCRIPTION	LOCATION
potential-dos-via-decompression-bomb	Detected a possible denial-of-service via a zip bomb attack.	bifrost/oracle/providers/digifinex.go:147:11 bifrost/oracle/providers/htx.go:156:11
no-direct-write-to-responsewriter	This bypasses HTML escaping that prevents cross-site scripting vulnerabilities.	bifrost/oracle/providers/mock/server.go:51:10

2.2 Go Sec

`go sec` (v2.22.7) identified 4 warnings from the codes in scope. They are listed as below.

WARNING	LOCATION
Errors unhandled	x/thorchain/querier.go:3559,3553,3512,3504

2.3 CodeQL

`codeQL` (v2.21.2) identified 4 warnings relevant to the scope. They are listed as below.

WARNING	LOCATION
Useless assignment to field	x/thorchain/types/type_observed_tx.go:203 x/thorchain/types/type_tss_metric.go: 19, 42
Writable file handle closed without error handling	config/config.go: 1180

3. TECHNICAL DETAILS OF SECURITY FINDINGS

This chapter provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the security findings.

#	SEVERITY	TITLE	STATUS
KS-THO-F-01	Medium	Unbuffered Channel Deadlock	Resolved
KS-THO-F-02	Low	Empty Result Not Checked	Resolved
KS-THO-F-03	Low	Check Ticker Never Stopped	Resolved
KS-THO-F-04	Low	Price Feed Deleted Before gRPC Confirmation	Resolved
KS-THO-F-05	Low	Return Missing	Resolved
KS-THO-F-06	Low	Weak Failover Logic	Resolved
KS-THO-F-07	Low	Iterator.Close Missing	Resolved

Findings overview.

3.1 KS-THO-F-01 Unbuffered Channel Deadlock

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

The oracle price collection process writes to an unbuffered channel without timeout protection. The consumer calls `processPriceFeedQueue` which invokes `AttestPriceFeed`, triggering `sendPriceFeedAttestationsToThornode` containing a blocking gRPC call. The `SendQuorumPriceFeedBatch` gRPC call lacks timeout protection and may block on network issues since price feeds are updated every 1 second by default.

3.2 KS-THO-F-02 Empty Result Not Checked

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The `FilterDeviations` function implements outlier detection using Median Absolute Deviation (MAD) to remove price data that deviates significantly from the median. However, it does not validate the edge case where all prices are filtered as outliers, resulting in an empty filtered map. The calling function `GetRate` also does not validate that the returned rates are non-empty before `ComputeVwap` is called.

3.3 KS-THO-F-03 Check Ticker Never Stopped

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

In the Start function, the `check` ticker is never stopped, causing a memory leak over time as tickers accumulate.

3.4 KS-THO-F-04 Price Feed Deleted Before gRPC Confirmation

Severity	Impact	Likelihood	Status
Low	Medium	Low	Resolved

Description

In the `sendPriceFeedAttestationsToThornode` function, price feed data is deleted from memory before confirming the gRPC call succeeded. If `SendQuorumPriceFeedBatch` fails, the data is permanently lost and no retry mechanism or recovery possible.

3.5 KS-THO-F-05 Return Missing

Severity	Impact	Likelihood	Status
Low	Medium	Low	Resolved

Description

In the `Poll` function, an error is logged but execution continues without return, potentially processing empty/nil ticker data. Also, a timestamp error is not handled properly. Similarly, heartbeat response failures are logged but ignored.

3.6 KS-THO-F-06 Weak Failover Logic

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The `httpRequest` function in `provider-base.go` is responsible for making HTTP requests to exchange APIs and implementing failover logic when multiple API endpoints are configured for a single provider. When a request fails and backup endpoints are available, the function is designed to rotate to the next endpoint in the `ApiEndpoints` array for subsequent requests. However, when the first endpoint (index 0) fails, the code finds index=0, checks if 0 == len-1 (false), and leaves index at 0, so it continues using the same failed endpoint instead of rotating to the backup.

3.7 KS-THO-F-07 Iterator.Close Missing

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

In the `handle` function, the iterator is created but there's no `iterator.Close` statement, which could cause resource leakage.

4. OBSERVATIONS

This chapter contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

#	SEVERITY	TITLE	STATUS
KS-THO-O-01	Informational	Best Secure Code Practice	Resolved
KS-THO-O-02	Informational	Potential Query DoS Attacks Through API Endpoints	Informational
KS-THO-O-03	Informational	Hybrid Oracle Failover Enhancement	Informational
KS-THO-O-04	Informational	Dead Code	Informational
KS-THO-O-05	Informational	Low Minimum Provider Limit	Informational
KS-THO-O-06	Informational	Handling Ticker ID Inconsistent	Informational

Observations overview.

4.1 KS-THO-O-01 Best Secure Code Practice

Description

- `RedistributePercentage` calculates a total of values but never uses it.
- This seems to be a typo.
- The input `pf` and `other` should not be `nil` but not checked.
- The input `number` should not be `nil` but not validated.
- Errors are logged but execution continues without return. In the end, the function returns no error.

4.2 KS-THO-O-02 Potential Query DoS Attacks Through API Endpoints

Description

The `QueryOraclePricesResponse` structure does not validate the `Prices` array size, which enables attackers to send malicious oracle price queries that trigger unbounded memory allocation. This exploits the protobuf unmarshaling process where memory is allocated via append operations. When unmarshaling fails on malformed entries, the partially allocated memory objects are never freed, creating a memory leak that accumulates with each attack attempt.

Additionally, the lack of rate limiting on gRPC oracle price queries allows attackers to flood validator nodes with unlimited query requests, potentially causing DoS attacks. Since oracle queries require no transaction fees, attackers can repeatedly exploit this vulnerability at zero cost to achieve complete network disruption.

4.3 KS-THO-O-03 Hybrid Oracle Failover Enhancement

Description

The current `HaltOracle` does not provide any fallback mechanism. When `HaltOracle` is activated, the enshrined oracle stops all price feed broadcasts, leaving ThorNode without any price data for cross-chain bridge operations, swaps, and liquidity management.

This gap would be addressed by implementing the automatic failover mechanism to external oracle feeds (for example, Chainlink or Pyth Network) during prolonged `HaltOracle` periods, ensuring continuous price data availability while maintaining operational security.

4.4 KS-THO-O-04 Dead Code

Description

In the `setPrice` function, the `MustMarshal` method either returns valid bytes or panics. Since it never returns `nil`, the deletion branch is unreachable dead code. The flawed logic may cause an unintended behaviour.

4.5 KS-THO-O-05 Low Minimum Provider Limit

Description

Enshrined oracle relies on the 18 different sources for the price feed of different pairs. However, the minimum 3 providers (`minProviders = 3`) is relatively low threshold.

4.6 KS-THO-O-06 Handling Ticker ID Inconsistent

Description

In `coinw.go`, ticker IDs are used for the pairs index in `PrepareConnection`, however in `HandleWsMessage`, ticker.Symbol is used as an index. Ticker processing may fail, causing missed price updates.

5. METHODOLOGY

For this engagement, Kudelski Security used a methodology that is described at a high level in this chapter. This is broken up into the following phases.

Kickoff
Ramp-up
Review
Report
Verify

5.1 Kickoff

The Kudelski Security Team set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting, we verified the scope of the engagement and discussed the project activities.

5.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps required for gaining familiarity with the codebase and technological innovations utilized.

5.3 Review

The review phase is where most of the work on the engagement was performed. In this phase we have analyzed the project for flaws and issues that could impact the security posture. The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools was used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Review

Kudelski Security Team reviewed the code within the project utilizing an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content, to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- authentication (*e.g.* A07:2021, CWE-306)
- authorization and access control (*e.g.* A01:2021, CWE-862)
- auditing and logging (*e.g.* A09:2021)
- injection and tampering (*e.g.* A03:2021, CWE-20)
- configuration issues (*e.g.* A05:2021, CWE-798)
- logic flaws (*e.g.* A04:2021, CWE-190)
- cryptography (*e.g.* A02:2021)

These categories incorporate common weaknesses and vulnerabilities such as the OWASP Top 10 and MITRE Top 25.

Smart Contracts

We reviewed the smart contracts, checking for additional specific issues that can arise such as:

- risk of centralization
- reentrancy
- (non)-adherence to existing standards
- unsafe arithmetic operations

5.4 Reporting

Kudelski Security delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project.

In the report we not only point out security issues identified but also observations for improvement. The findings are categorized into several buckets, according to their overall severity: **Critical**, **High**, **Medium**, **Low**.

Observations are considered to be **Informational**. Observations can also consist of code review, issues identified during the code review that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

5.5 Verify

After the preliminary findings have been delivered, we verify the fixes applied by the Client. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase is the final report with any mitigated findings noted.

6. VULNERABILITY SCORING SYSTEM

Kudelski Security utilizes a custom approach when computing the vulnerability score, based primarily on the **Impact** of the vulnerability and **Likelihood** of an attack.

Each metric is assigned a ranking of either low, medium or high, based on the criteria defined below. The overall severity score is then computed as described in the next section.

Severity

Severity is the overall score of the finding, weakness or vulnerability as computed from Impact and Likelihood. Other factors, such as availability of tools and exploits, number of instances of the vulnerability and ease of exploitation might also be taken into account when computing the final severity score.

IMPACT / LIKELIHOOD	LOW	MEDIUM	HIGH
HIGH	MEDIUM	HIGH	HIGH
MEDIUM	LOW	MEDIUM	HIGH
LOW	LOW	LOW	MEDIUM

Compute overall severity from Impact and Likelihood. The final severity factor might vary depending on a project's specific context and risk factors.

- **Critical** The identified issue may be immediately exploitable, causing a strong and major negative impact system-wide. They should be urgently remediated or mitigated.
- **High** The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
- **Medium** The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
- **Low** The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
- **Informational** findings are best practice steps that can be used to harden the application and improve processes. Informational findings are not assigned a severity score and are classified as Informational instead.

Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

- **High** The vulnerability has a severe effect on the company and systems or has an effect within one of the primary areas of concern noted by the client.
- **Medium** It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
- **Low** There is little to no effect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, difficulty of exploitation and institutional knowledge.

- **High** It is extremely likely that this vulnerability will be discovered and abused.
- **Medium** It is likely that this vulnerability will be discovered and abused by a skilled attacker.
- **Low** It is unlikely that this vulnerability will be discovered or abused when discovered.

7. CONCLUSION

The objective of this code review was to evaluate the overall security of the code base and identify any vulnerabilities that would put the product at risk.

The Kudelski Security Team identified 7 security issues: 1 medium risk, and 6 low risks. On average, the effort needed to mitigate these risks is estimated as `low`.

In order to mitigate the risks posed by this engagement's findings, the Kudelski Security Team recommends applying the following best practices:

- Ensure the channel deadlock never happens
- Edge cases need to be considered
- Resource leakage should be avoided

The Thorchain team addressed all these vulnerabilities in the follow-up revision of the codebase.

Kudelski Security remains at your disposal should you have any questions or need further assistance.

Kudelski Security would like to thank Thorchain for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.

8. REFERENCES

- https://coinfomania.com/thorchain-introduces-enshrined-oracles-to-strengthen-price-accuracy-and-protocol-integrity/
- https://x.com/thorchain/status/1932690942093410542?s=46
- https://gitlab.com/thorchain/thornode/-/blob/develop/docs/concepts/oracle.md?ref_type=heads
- https://x.com/THORChain/status/1958263114601820162
- https://medium.com/@nfett/pros-and-cons-of-enshrined-oracles-6b15c46e2410