Security Audit Report

The Rig

v1.0

July 1, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged to perform a security audit of The Rig.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/Rig-Labs/the-rig
Commit	`26bcde7b614e78f59dc9970f4f813a3fe85749f9`
Scope	The scope is restricted to the contracts defined in: • `ignition/contracts/price_feed` • `ignition/contracts/rig` • `ethereum/src`, excluding the `vendor` directory
Fixes verified at commit	`df17d40f524c5c1d59eeab6ad06fdda9c0b93609`

	Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The Rig is a liquid staking protocol for the Fuel Network that enables users to stake their FUEL tokens while maintaining liquidity through a derivative token called stFUEL (Liquid Staked FUEL).

The protocol operates across both Ethereum (L1), Ignition (L2), and Sequencer chains, leveraging cross-chain infrastructure to provide non-custodial staking services.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	High	The client provided detailed documentation outlining the specifications of the intended protocol behavior.
Test coverage	Low-Medium	`forge coverage` reports a test coverage of `40.39%`. It is not possible to compute test coverage for Sway contracts.

Summary of Findings

No	Description	Severity	Status
1	Missing price staleness check may lead to incorrect stFUEL minting	Major	Resolved
2	Strict deviation threshold prevents timely price corrections and enables arbitrage risks	Major	Acknowledged
3	Premature round finalization undermines aggregation window and enables timing attacks	Major	Acknowledged
4	Insufficient stFUEL minting may lead to a denial of service during stFUEL claiming	Minor	Acknowledged
5	Missing slippage protection for deposits	Minor	Resolved
6	Missing address validation	Minor	Resolved
7	Inconsistent invariant enforcement enables misconfiguration in the price submission logic	Minor	Resolved
8	Integer underflow in the deviation threshold enables DoS for price submissions	Minor	Resolved
9	Lack of initial price validation enables permanent DoS in the price feed	Minor	Resolved
10	Inefficient median calculation enables gas-exhaustion DoS in price aggregation	Minor	Acknowledged
11	Incorrect round end timestamp	Minor	Resolved
12	Centralization risks	Minor	Acknowledged
13	Incomplete interface definition	Informational	Resolved
14	Mark an expired price feed round as failed, regardless of whether the minimum submission count is met	Informational	Resolved
15	Missing price feed configuration variable validation	Informational	Resolved
16	The `withdraw_to_l1_rig` function mistakenly refers to stFUEL instead of FUEL tokens	Informational	Resolved
17	Lack of event emission for the `setRigTreasury` function	Informational	Resolved

18	Missing interface validation may lead to misconfiguration or runtime errors	Informational	Acknowledged
19	Missing event emission on pause	Informational	Resolved

Detailed Findings

1. Missing price staleness check may lead to incorrect stFUEL minting

Severity: Major

In `ignition/contracts/rig/src/main.sw:532-535`, the `_get_price` function fetches a price from an external `PriceFeed` contract without validating the freshness or timestamp of the returned price.

In scenarios where submitters fail to complete rounds, the system may operate on outdated or stale price data, potentially resulting in incorrect minting of stFUEL and economic imbalances.

Recommendation

We recommend modifying the `get_price` function in the `price_feed` contract to return additional metadata. Specifically, the timestamp or `latest_updated_round` associated with the price value.

This would allow consumers of the `_get_price` function, such as the one in `ignition/contracts/rig/src/main.sw:532-535`, to verify whether the price data is fresh or stale before using it for critical operations like minting stFUEL.

Status: Resolved

2. Strict deviation threshold prevents timely price corrections and enables arbitrage risks

Severity: Major

In `ignition/contracts/price_feed/src/main.sw:161-167`, the `submit_price` function uses `current_price` value as the reference for validating submitted prices.

The deviation threshold is calculated as `current_price ± max_deviation`, where `max_deviation` is derived proportionally from the configured `max_deviation_submission`.

This logic introduces multiple issues:

1. If the initial `current_price` is set very low (e.g., 1), and the true market price is significantly higher (e.g., 2), submitters are blocked from correcting the price due to the deviation limit.

2. In cases where the market price moves rapidly (e.g., during periods of high volatility in secondary markets), the deviation threshold may prevent updates that accurately reflect the current price. This results in stale pricing, which can be exploited for arbitrage, particularly when the stFUEL/FEUL rate on Ethereum diverges from the on-chain price used by the Rig.

3. The `current_price` is only updated when `_try_aggregate_round` successfully completes and commits a new median value. During consecutive round failures, caused by timeouts or insufficient submissions, `current_price` remains stale. This stagnation creates a cascading failure scenario: as market conditions change, legitimate price submissions from oracles begin to fall outside the fixed deviation boundaries, triggering rejections via the `PriceOutsideDeviationThreshold` error. These rejections prevent round completion, further freezing the `current_price`, and reinforcing the invalid state. Over time, this can render the contract permanently incapable of accepting valid price data, especially during periods of high volatility when reliable price feeds are most critical.

Recommendation

We recommend relaxing this deviation check and allowing the price to be updated regardless of its deviation from the current value. Instead of rejecting the update, the responsibility for validating deviations and assessing metadata such as staleness should be delegated to the consumer contracts (e.g., the rig). This approach ensures that prices remain up-to-date on-chain while enabling downstream consumers to apply their own validation logic based on their specific risk tolerance and requirements.

Status: Acknowledged

The client states that the price feed reflects the redemption rate of stFUEL to FUEL and is calculated based on the ratio between the total amount of FUEL in the system and the total amount of minted stFUEL. This rate changes only in the event of validator slashing or through inflation, which is approximately 30% APY.

A strict deviation threshold is enforced due to the centralized nature of the price feed operators and the potential for unforeseen bugs. Calculating the total amount of FUEL in the system is complex, as it requires tracking two bridges (associated with off-chain withdrawals) and operations across three separate chains.

The client further states that they have implemented monitoring mechanisms for submitted prices. In the event of rapid changes, such as multiple validators being slashed, they will receive alerts and can temporarily adjust the maximum deviation configuration as needed.

3. Premature round finalization undermines aggregation window and enables timing attacks

Severity: Major

In `ignition/contracts/price_feed/src/main.sw:368-371`, the `_try_aggregate_round` function checks if the number of submissions is below `config.min_submissions`, and if not, immediately proceeds to finalize the round and calculate the median.

This behavior causes the round to complete as soon as the minimum number of submissions is received, regardless of whether the configured `round_time` has elapsed.

As a result, the `min_submissions` parameter effectively becomes a maximum cap. For example, if a round is configured with `min_submissions` equal to `3` and round_time equal to 1 hour, the round will finalize instantly once three oracles submit prices, potentially within seconds.

This excludes all subsequent submissions from other participants, reducing data diversity and quality. It also introduces a race condition that incentivizes timing manipulation, allowing early submitters to influence the outcome while denying participation to others.

This behavior is contrary to the standard expectations for decentralized oracle systems, which are designed to aggregate submissions over the entire collection window to maximize robustness and resistance to manipulation.

Recommendation

We recommend decoupling the round finalization logic from the `min_submissions` threshold. The round should only be finalized once both conditions are met: the minimum number of submissions has been received and the full `round_time` has elapsed.

Status: Acknowledged

The client states that they prefer rounds to end early in the event of validator slashing. Since the mechanism serves as a redemption oracle rather than a market oracle, it is important for rounds to conclude promptly if significant events, such as slashing, occur early in the round (e.g., within the first five minutes).

They further explain that operators are configured to submit prices five minutes before the round ends, provided there are no major off-chain deviations. In cases where such deviations are detected, prices are submitted earlier.

4. Insufficient stFUEL minting may lead to a denial of service during stFUEL claiming

Severity: Minor

In `ignition/contracts/rig/src/main.sw:251-254`, the `claim` function calculates the user's claimable amount as their original deposit plus rewards, then attempts to transfer this total in stFUEL.

However, the contract only mints stFUEL equivalent to the initial deposit amounts during initialization and does not account for future rewards. Since no mechanism exists to automatically mint or top up stFUEL for reward payouts, the contract may run out of stFUEL, causing the `claim` function to fail. This introduces a denial-of-service risk for users attempting to claim their eligible stFUEL tokens.

Recommendation

We recommend preemptively minting stFUEL during initialization for reward distribution.

Status: Acknowledged

5. Missing slippage protection for deposits

Severity: Minor

In `ignition/contracts/rig/src/main.sw:182-187`, the `deposit` function lacks a mechanism to enforce slippage protection. Specifically, it does not accept a `min_output_amount` or similar parameter that would allow users to specify a minimum acceptable return for their deposit.

This omission exposes users to the risk of front-running or price manipulation between the time a transaction is submitted and when it is executed. Without a slippage check, users may receive significantly less value than expected if prices shift unfavorably during this window, especially in volatile market conditions or under adversarial execution environments.

Recommendation

We recommend extending the `deposit` function to include a user-specified minimum output amount parameter.

Status: Resolved

6. Missing address validation

Severity: Minor

In `ethereum/src/Rig.sol:60-61` and `ethereum/src/Rig.sol:73-75`, the `initialize` and `setRigTreasury` functions store addresses provided as input without performing any validation.

Specifically, the `initialize` function sets the `sequencerInterface` and `fuelToken`, while `setRigTreasury` assigns the `treasury` address.

If any of these inputs are the zero address (0x0), subsequent contract operations, such as token transfers or sequencer interactions, will fail.

Recommendation

We recommend validating addresses before storing them in the contract.

Status: Resolved

7. Inconsistent invariant enforcement enables misconfiguration in the price submission logic

Severity: Minor

In `ignition/contracts/price_feed/src/main.sw:308-356`, there is an inconsistency between the `_set_config` and `_set_price_submitter` functions.

The `_set_config` function enforces the invariant that `config.min_submissions` must not exceed the total number of active submitters, ensuring that price finalization is possible.

However, `_set_price_submitter` lacks a reciprocal check to maintain this invariant when modifying the active submitter count.

As a result, a submitter could be deactivated after setting a valid config, reducing `total_active_submitters` below `min_submissions` and rendering the system unable to finalize a price round.

Recommendation

We recommend enforcing the same invariant within `_set_price_submitter` to prevent the active submitter count from falling below the configured `min_submissions`.

Status: Resolved

8. Integer underflow in the deviation threshold enables DoS for price submissions

Severity: Minor

In `ignition/contracts/price_feed/src/main.sw:164-165`, the `submit_price` function calculates an acceptable deviation range based on `current_price` and `max_deviation_submission`.

However, when `max_deviation_submission` exceeds `current_price`, the subtraction between `current_price` and `max_deviation` results in an integer underflow due to the use of unsigned 64-bit integers.

For instance, if the current price is `100` units and the `max_deviation_submission` is set to `150,000` basis points (equivalent to 150%), the computed deviation becomes `150`. This leads to a lower bound calculation of `100` minus `150`, which yields `-50`.

Since negative values are invalid for `u64`, this causes a runtime panic under Sway's safe math enforcement, rendering the contract unable to accept any further price submissions and effectively triggering a denial of service (DoS).

Recommendation

We recommend enforcing an upper limit on `max_deviation_submission`, ensuring it remains below `100,000` (i.e., less than 100%).

Alternatively, implementing saturating arithmetic or enforcing a minimum lower bound of `1` would prevent underflow conditions and maintain submission availability.

Status: Resolved

9. Lack of initial price validation enables permanent DoS in the price feed

Severity: Minor

In `ignition/contracts/price_feed/src/main.sw:123-126`, the `initialize` function writes the `initial_price` directly to `storage.current_price` without any validation.

However, if `initial_price` is set to zero, the contract enters a permanently disabled state. When a user subsequently invokes `submit_price`, the stored value of zero is read into `current_price`, resulting in a deviation calculation of zero, regardless of `max_deviation_submission`. This causes the validation logic to require `price` equal to `0`, effectively rejecting all non-zero price submissions.

The aggregation process computes and stores a median price of zero via `_try_aggregate_round` and `_calculate_median`, reinforcing the invalid state. This

creates a self-sustaining condition where only zero-priced submissions are accepted, blocking any future meaningful price updates.

Recommendation

We recommend enforcing a strict check during initialization to reject `initial_price` values equal to zero.

Status: Resolved

10. Inefficient median calculation enables gas-exhaustion DoS in price aggregation

Severity: Minor

In `ignition/contracts/price_feed/src/main.sw:423-450`, the `_calculate_median` function implements a sort algorithm with $O(n^2)$ time complexity to sort submitted prices.

This algorithm performs nested iterations over the `sorted_prices` vector, incurring multiple storage reads and writes for each comparison and swap. Given that each of these operations consumes gas under the Sway execution model, the total gas cost scales quadratically with the number of price submissions.

Consequently, depending on the cardinality of submitters per round, the execution could run out of gas during median calculation in the `_try_aggregate_round` function.

This would cause round aggregation to fail, preventing legitimate price updates and potentially rendering the oracle system non-functional.

Recommendation

We recommend replacing the sort algorithm with a more gas-efficient one.

Status: Acknowledged

11. Incorrect round end timestamp

Severity: Minor

In `ignition/contracts/price_feed/src/main.sw:360-420`, the `_try_aggregate_round` function emits a `RoundStarted` event to indicate the initiation of a new price aggregation round.

However, the `round_end` field in the event is incorrectly set to the end timestamp of the previous round rather than computing the correct end time for the new round.

This misrepresents the duration of the newly started round in emitted logs, potentially misleading off-chain systems, monitoring tools, or oracles that rely on event data for synchronization or scheduling.

Recommendation

We recommend updating the `round_end` value in the `RoundStarted` event emission to reflect the end of the new round.

Status: Resolved

12. Centralization risks

Severity: Minor

The Rig contract, defined in `ethereum/src/Rig.sol`, is designed with a high degree of centralization.

Admin and operators granted the `OPERATOR_ROLE`, are authorized to perform critical actions such as deposits, delegations, and fund transfers without oversight.

Specifically, the `delegate` function allows the operator to unilaterally choose validators, and the `transfer` function enables the operator to withdraw any amount of funds without rate limits, validation checks, or multi-signature enforcement.

In addition, the `end_round` function, responsible for marking expired rounds and starting new ones, is implemented as a permissioned function restricted to submitters. If submitters fail to call `end_round`, either due to inactivity, network issues, or collusion, the oracle halts.

This centralization creates a significant single point of control and risk, which could be exploited maliciously or result in unintentional mismanagement of funds.

Recommendation

We recommend introducing governance mechanisms or multi-signature requirements for critical operations such as fund transfers and validator selection.

Additionally, implementing safeguards such as withdrawal limits or time locks can reduce the risk associated with a centralized operator role.

Status: Acknowledged

13. Incomplete interface definition

Severity: Informational

In `ethereum/src/interfaces/IRig.sol:6-45`, the `IRig` interface omits the declaration of the `depositAndDelegate` function, which is defined and implemented in `ethereum/src/Rig.sol`.

This inconsistency breaks interface completeness and may lead to integration issues for external systems or contracts relying on the interface to interact with the full Rig contract API.

The missing function can result in unexpected behavior during dependency injection, contract upgrades, or tooling that depends on accurate ABI representation.

Recommendation

We recommend updating the `IRig` interface to include the `depositAndDelegate` function signature.

Status: Resolved

14. Mark an expired price feed round as failed, regardless of whether the minimum submission count is met

Severity: Informational

In `ignition/contracts/price_feed/src/main.sw:368-370`, the `_try_aggregate_round` function attempts to aggregate the submitted prices for a given round into a single, final price.

It first checks if there are enough submissions to meet the minimum before handling an expired round. If a round expires without enough submissions, it is not immediately marked as failed. Instead, it needs manual intervention via an extra `end_round` function call.

Recommendation

We recommend first checking whether the round has expired and marking the round as failed, regardless of the number of price submissions.

Status: Resolved

15. Missing price feed configuration variable validation

Severity: Informational

In `ignition/contracts/price_feed/src/main.sw:308-326`, the `_set_config` function, which is internally called by the owner-callable `set_config` function and during

initialization, allows changing the `min_submissions`, `round_time`, and `max_deviation_submission` configuration variables.

While basic validation ensures that those values are larger than zero, it is recommended to restrict the values to a reasonable range.

Specifically,

- A maximum `round_time` (e.g., 1 day)
- A maximum `max_deviation_submission` (e.g., 10%)

Recommendation

We recommend adding the validations mentioned above to the `_set_config` function.

Status: Resolved

16. The `withdraw_to_l1_rig` function mistakenly refers to stFUEL instead of FUEL tokens

Severity: Informational

In `ignition/contracts/rig/src/main.sw:267,290`, the `withdraw_to_l1_rig` function, which allows withdrawing FUEL tokens to the L1, incorrectly refers to stFUEL in the code comments, confusing the code's intent.

Recommendation

We recommend changing both instances to mention FUEL instead of stFUEL.

Status: Resolved

17. Lack of event emission for the `setRigTreasury` function

Severity: Informational

In `ethereum/src/Rig.sol:73-75`, the `setRigTreasury` function sets the Rig treasury address, which receives the collected protocol fees.

However, the function does not emit an event about this state change, thereby preventing off-chain tools from monitoring the functionality.

Recommendation

We recommend emitting an event.

Status: Resolved

18. Missing interface validation may lead to misconfiguration or runtime errors

Severity: Informational

In `ignition/contracts/rig/src/main.sw:309-316`, the `set_price_feed_contract` function allows the owner to set a new price feed contract address, but it does not perform any validation to ensure that the provided contract actually conforms to the expected interface.

Without verifying that the new contract implements required methods (such as `get_price`), this setup may result in misconfigurations or runtime errors during price retrieval operations, potentially affecting the correctness of the contract.

Recommendation

We recommend adding a validation step within `set_price_feed_contract()` to ensure the provided contract implements the expected interface, such as calling `get_price()` in a no-op fashion to confirm it responds correctly

Status: Acknowledged

19. Missing event emission on pause

Severity: Informational

In `ignition/contracts/rig/src/main.sw:351-360`, the pause function updates the contract's paused state but fails to emit a `PausedUpdateEvent` to signal this change.

In contrast, the corresponding `unpause` function correctly emits an event, ensuring state changes are visible to off-chain systems and audit tools.

Recommendation

We recommend adding a `PausedUpdateEvent` emission within the `pause` function.

Status: Resolved