Security Audit Report

The Rig Updates

v1.1

October 1, 2025

Table of Contents

Table of Contents **2**
License **3**
Disclaimer **4**
Introduction **5**
Purpose of This Report 5
Codebase Submitted for the Audit 5
Methodology 7
Functionality Overview 7
How to Read This Report **8**
Code Quality Criteria **9**
Summary of Findings **10**
Detailed Findings **11**
1. Incorrect stFUEL to FUEL ratio calculation 11
2. Event emission flaw leading to a misleading deposited amount in claim 11
3. The full amount of FUEL to be staked cannot be withdrawn to L1 12
4. Minimum withdrawal check incorrectly compares stFUEL with FUEL amount 12
5. Missing input validation for stfuel_asset_id 13
6. Missing validation for fuel_unbonded 13
7. Missing supply reduction on burns causes inflated total_supply tracking 14
8. Missing initialization check in process_message may allow execution in an uninitialized state 14
9. Validation order can be improved to prevent wasted execution 15
10. Balance increase check ordering can be improved 15
11. Redundant minimum received check can be avoided 15
12. Misleading fee configuration comment may cause incorrect assumptions 16
13. Early return can reduce unnecessary computation and gas 16
14. Missing validation allows zero-value withdrawals 17

License

Disclaimer

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 updates to 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 `ignition/contracts/staking_migration` directory.
Fixes verified at commit	`daa19d427327314cd85fae169519c7e1aaa415e3` 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.

Repository	https://github.com/Rig-Labs/the-rig
Commit	`82c3a289b302de8df13098f6860dd3102e8028fc`
Scope	The scope is restricted to the changes applied in commit `82c3a289b302de8df13098f6860dd3102e8028fc`
Fixes verified at commit	`daa19d427327314cd85fae169519c7e1aaa415e3` 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 `40.78%`. It is not possible to compute test coverage for Sway contracts.

Summary of Findings

No	Description	Severity	Status
1	Incorrect stFUEL to FUEL ratio calculation	Minor	Resolved
2	Event emission flaw leading to a misleading deposited amount in `claim`	Minor	Resolved
3	The full amount of FUEL to be staked cannot be withdrawn to L1	Minor	Resolved
4	Minimum withdrawal check incorrectly compares stFUEL with FUEL amount	Minor	Resolved
5	Missing input validation for `stfuel_asset_id`	Minor	Acknowledged
6	Missing validation for `fuel_unbonded`	Minor	Resolved
7	Missing supply reduction on burns causes inflated `total_supply` tracking	Minor	Resolved
8	Missing initialization check in `process_message` may allow execution in an uninitialized state	Minor	Resolved
9	Validation order can be improved to prevent wasted execution	Informational	Resolved
10	Balance increase check ordering can be improved	Informational	Resolved
11	Redundant minimum received check can be avoided	Informational	Resolved
12	Misleading fee configuration comment may cause incorrect assumptions	Informational	Resolved
13	Early return can reduce unnecessary computation and gas	Informational	Acknowledged
14	Missing validation allows zero-value withdrawals	Informational	Resolved

Detailed Findings

1. Incorrect stFUEL to FUEL ratio calculation

Severity: Minor

In `ignition/contracts/staking_migration/src/main.sw:117-123`, the `initial_setup` logic calculates the stFUEL to FUEL ratio using the post-deposit stFUEL balance rather than the delta of newly minted stFUEL.

If the contract already holds stFUEL before initialization, the ratio is artificially inflated, resulting in an incorrect stFUEL to FUEL ratio.

Recommendation

We recommend computing the stFUEL to FUEL ratio by capturing the minted delta rather than relying on the post-deposit balance.

Status: Resolved

2. Event emission flaw leading to a misleading deposited amount in `claim`

Severity: Minor

In `ignition/contracts/staking_migration/src/main.sw:130-156`, the `claim` function sets the user's deposit to zero before the `ClaimedStFuelEvent` is logged.

As a result, the event always reports `amount_deposited` equal to zero, regardless of the actual deposited value prior to the claim. This creates misleading on-chain logs, which could hinder auditors, monitoring tools, and downstream applications relying on accurate event data.

Recommendation

We recommend updating the order of operations so that the event is logged before setting the deposit to zero. Alternatively, store the original deposit amount in a temporary variable and use it in the event emission.

Status: Resolved

3. The full amount of FUEL to be staked cannot be withdrawn to L1

Severity: Minor

In `ignition/contracts/rig_v2/src/main.sw:247-284`, the `withdraw_to_l1_rig` function allows an operator to withdraw the to-be-staked FUEL tokens to the L1 Rig address.

However, the function validates in lines `265-268` that the `amount` to be withdrawn is strictly less than the `total_to_be_staked` balance.

As a result, it is not possible for the operator to withdraw the entire `total_to_be_staked` amount, leaving a small amount of funds in the contract that cannot be staked.

Recommendation

We recommend changing the strict inequality > to >= to allow the full FUEL amount to be withdrawn.

Status: Resolved

4. Minimum withdrawal check incorrectly compares stFUEL with FUEL amount

Severity: Minor

In `request_withdrawal`, defined in `ignition/contracts/rig_v2/src/main.sw:403-407`, the validation for the minimum withdrawal amount incorrectly compares values of different asset denominations. The check compares `st_fuel_amount`, which is denominated in stFUEL, with the `min_fuel_withdrawal` configuration value, which is denominated in FUEL.

As the exchange rate between stFUEL and FUEL is not guaranteed to be 1:1, this direct comparison is flawed. Depending on the relative values of the two assets, this could either prevent users from making valid withdrawals that are above the minimum threshold or allow withdrawals that are effectively below the intended minimum FUEL amount.

Recommendation

We recommend removing the redundant and incorrect check on `st_fuel_amount` in lines `403-407`. The existing check on `fuel_to_withdraw` is sufficient for enforcing the minimum withdrawal amount in FUEL.

If a separate minimum withdrawal threshold for stFUEL is desired, we recommend introducing a new configuration variable, such as `min_st_fuel_withdrawal`, and using it for comparison with `st_fuel_amount`.

Status: Resolved

5. Missing input validation for `stfuel_asset_id`

Severity: Minor

In `ignition/contracts/staking_migration/src/main.sw:76-79`, the contract does not validate `stfuel_asset_id` before use.

If an invalid or unintended token address is provided, deposits into the Rig contract may succeed, but the migration contract will mint `stfuel_tokens` to the wrong asset.

During claims, the migration logic would then attempt to transfer incorrect tokens, causing users' migrated funds to become irretrievable.

This creates a risk of permanently stuck assets and failed user claims.

Recommendation

We recommend validating the `stfuel_asset_id`.

Status: Acknowledged

6. Missing validation for `fuel_unbonded`

Severity: Minor

In `ignition/contracts/rig_v2/src/main.sw:615`, the `batch_unbonded` function directly assigns the `fuel_unbonded` value provided by the operator without validation.

A compromised or malicious operator could set `fuel_unbonded` to an amount greater than the actual unbonded funds, inflating withdrawal entitlements.

This breaks accounting consistency, potentially leading to skewed payouts.

Recommendation

We recommend enforcing strict validation on the `fuel_unbonded` value by cross-checking it against protocol state and expected unbonding amounts before updating storage. Any operator-provided value should be bounded to ensure it cannot exceed the actual unbonded balance.

Status: Resolved

7. Missing supply reduction on burns causes inflated `total_supply` tracking

Severity: Minor

In `ignition/contracts/rig_v2/src/main.sw:225`, the contract correctly increments `storage::staking::v1.total_supply` when minting stFUEL but fails to decrement it during burns, such as instant withdrawals or batch finalizations.

As a result, `SRC20.total_supply` drifts upward over time and no longer reflects the true circulating supply.

This inconsistency undermines supply integrity, misleads monitoring tools, and may lead to failures in systems that rely on accurate supply data for validation, accounting, or auditing purposes.

Recommendation

We recommend updating the logic to ensure `total_supply` is decremented whenever stFUEL is burned, maintaining parity with the circulating token supply.

Status: Resolved

8. Missing initialization check in `process_message` may allow execution in an uninitialized state

Severity: Minor

In `ignition/contracts/rig_v2/src/main.sw:927`, the `process_message` function enforces reentrancy protection, pause state, and operator authorization, but does not validate that the contract has been properly initialized.

Without an `_is_initialized` check, the function could be executed in an uninitialized state, leading to undefined behavior, inconsistent storage, or bypass of expected setup invariants.

Recommendation

We recommend adding an explicit `_is_initialized` requirement at the start of `process_message` to ensure the function cannot be invoked before the contract is properly set up.

Status: Resolved

9. Validation order can be improved to prevent wasted execution

Severity: Informational

In `ignition/contracts/staking_migration/src/main.sw:90`, the contract performs a `require` check to ensure `rig_contract` is not the zero address.

However, the validation occurs after other logic in the function, rather than at the beginning.

Early validation ensures invalid input is caught sooner, preventing wasted execution and reducing unnecessary gas usage.

Recommendation

We recommend placing the `require(rig_contract != ContractId::zero(), …)` at the start of the function to validate the configuration before any further processing.

Status: Resolved

10. Balance increase check ordering can be improved

Severity: Informational

In `ignition/contracts/staking_migration/src/main.sw:106-109`, the code computes `st_fuel_received` by subtracting balances before verifying that `after > before`.

The requirement should be placed before the subtraction to prevent potential underflow or wasted gas in case of failure.

Recommendation

We recommend moving the `require(st_fuel_balance_after > st_fuel_balance_before, …)` before the subtraction to ensure validity is checked prior to computing `st_fuel_received`.

Status: Resolved

11. Redundant minimum received check can be avoided

Severity: Informational

In `ignition/contracts/staking_migration/src/main.sw:110-111`, the contract enforces `require(st_fuel_received >= min_st_fuel_received, …)` after calling `rig.deposit()`.

This check is redundant because the same protection can be applied directly within the `rig.deposit()` call. By passing `min_st_fuel_received` as a parameter to the deposit, slippage and dust handling are delegated to the RIG contract, eliminating the need for a second validation and saving gas.

Recommendation

We recommend passing `min_st_fuel_received` into `rig.deposit()` instead of performing a separate require check.

Status: Resolved

12. Misleading fee configuration comment may cause incorrect assumptions

Severity: Informational

In `ignition/contracts/rig_v2/src/main.sw:136-138`, the `withdrawal_fee` parameter is set to `500`, which corresponds to `5%` when expressed in basis points (bps).

However, the accompanying comment incorrectly states `0.05%`.

This inconsistency does not alter execution but can mislead developers, auditors, or integrators into believing a significantly smaller fee is applied.

Recommendation

We recommend correcting the comment to accurately reflect the configured value of `5%`, or adjusting the code to align with the intended fee of `0.05%`.

Status: Resolved

13. Early return can reduce unnecessary computation and gas

Severity: Informational

In `ignition/contracts/staking_migration/src/main.sw:84-87`, the function retrieves `contract_fuel_balance` and `v1_total_deposited` before proceeding with migration logic.

When `v1_total_deposited` is zero, the migration performs unnecessary checks and computations, even though there is nothing to migrate.

Recommendation

We recommend returning early if `v1_total_deposited == 0` to save gas and avoid redundant processing.

Status: Acknowledged

14. Missing validation allows zero-value withdrawals

Severity: Informational

In `ignition/contracts/rig_v2/src/main.sw:303,370`, the contract retrieves the withdrawal amount via the `msg_amount` function but does not enforce that the value is greater than zero.

This omission allows zero-value requests, which, while not directly harmful, introduce unnecessary gas usage and computations.

Recommendation

We recommend ensuring that the provided token amount is non-zero.

Status: Resolved