



SMART CONTRACT SECURITY AUDIT

Rig Labs

USDM Oracle Contract

CLIENT

Rig Labs

PLATFORM

Fuel

LANGUAGE

Sway

REVIEW

October 2025

DOCUMENT

Document information

Project	Rig Labs, USDM Oracle Contract
Client	Rig Labs
Platform	Fuel
Language	Sway
Repository	Rig-Labs/Moor: contracts/oracle-contract/src/main.sw
Commit	Not recorded in the original report
Methods	Manual code review, static analysis, best practices check
Assessed	October 2025

Document history

VERSION	DATE	CHANGE
1.0	October 2025	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	H-01 Inconsistent Timestamp Comparison Between Fuel TAI64 and Oracle UNIX Formats	8
	M-01 Recursive transfer_ownership Call Causes Guaranteed Revert	9
	L-01 get_price Uses Pyth Price Without Validating Confidence Interval	10
	I-01 Missing Zero Address Validation in transfer_ownership	10
	I-02 get_price Always Prioritizes Stork Oracle, Ignoring Potentially Fresher Data	11
A	Appendix: vulnerability checklist	12
	Disclaimer	13

Executive summary

About Rig Labs

Rig Labs' USDM oracle contract on Fuel aggregates prices from Stork, Pyth and Redstone, with staleness checks and a last-good-price fallback.

Summary of the review

Safe Edges reviewed the oracle contract behind Rig Labs' USDM on Fuel in October 2025. Five issues were identified: one High, one Medium, one Low and two Informational. The High issue was a staleness check that compared Fuel's TAI64 block timestamps directly with UNIX oracle timestamps, and the Medium issue was a transfer_ownership function that recursively called itself and always reverted. All five were fixed.

Findings by severity



Findings by status

STATUS	ISSUES
● Resolved	5

Scope

Repository	Rig-Labs/Moor: contracts/oracle-contract/src/main.sw
Commit	Not recorded in the original report
Platform	Fuel
Language	Sway

IN-SCOPE FILES

```
contracts/oracle-contract/src/main.sw
```

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Manual code review, static analysis, best practices check.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
H-01	Inconsistent Timestamp Comparison Between Fuel TAI64 and Oracle UNIX Formats	High	● Resolved
M-01	Recursive transfer_ownership Call Causes Guaranteed Revert	Medium	● Resolved
L-01	get_price Uses Pyth Price Without Validating Confidence Interval	Low	● Resolved
I-01	Missing Zero Address Validation in transfer_ownership	Informational	● Resolved
I-02	get_price Always Prioritizes Stork Oracle, Ignoring Potentially Fresher Data	Informational	● Resolved

Detailed findings

H-01 Inconsistent Timestamp Comparison Between Fuel TAI64 and Oracle UNIX Formats

Severity **High** Status ● Resolved

DESCRIPTION

The `get_price` function performs a staleness check using the condition

```
fn set_debug_timestamp(timestamp: u64) {
    only_owner();
    // Allow setting a custom timestamp for testing, but only in debug mode
    require(DEBUG, OracleError::DebugNotEnabled);
    @> storage.debug_timestamp.write(timestamp);
}

#[storage(read, write)]
fn get_price() -> u64 {
    // Determine the current timestamp based on debug mode
    let current_time = match DEBUG {
        true => storage.debug_timestamp.read(),
    @> false => timestamp(),
    };
    @> if current_time <= stork_timestamp + TIMEOUT {}
    @> if current_time <= pyth_price.publish_time + TIMEOUT {}
    @> if current_time <= redstone_timestamp + TIMEOUT {}
```

Here, `current_time` is obtained from the Fuel networks block timestamp (`timestamp()`), which is expressed in TAI64 format. In contrast, oracle-provided timestamps (from Pyth, Redstone, and Stork) use the UNIX timestamp format (seconds since 1970).

Since TAI64 and UNIX timestamps are fundamentally different representations, comparing them directly will yield meaningless or incorrect results. This renders the staleness validation ineffective, potentially causing stale oracle data to be accepted as fresh, or fresh data to be rejected as stale.

IMPACT

This issue would cause inconsistent behaviour and incorrect staleness check.

RECOMMENDED MITIGATION

We recommend using the Fuel SDK Date conversion utilities to handle timestamp conversions. Alternatively, you may implement the necessary conversions manually, ensuring both TAI64 and UNIX timestamps are properly aligned.

M-01 Recursive transfer_ownership Call Causes Guaranteed RevertSeverity **Medium** Status ● Resolved**DESCRIPTION**

The contract defines a public function `transfer_ownership` which is intended to delegate ownership transfer to the `sway_libs::ownership::transfer_ownership` function. However, because both share the same name, the locally defined function shadows the imported one.

Sway allows defining a local function with the same name as an imported one, but if the call `transfer_ownership(new_owner)` is not disambiguated to explicitly reference the imported function, it resolves to the local function itself. This causes infinite recursion, eventually consuming all available gas and reverting the transaction

```
use sway_libs::ownership::{
    _owner,
    initialize_ownership,
    transfer_ownership,
    only_owner,
};

#[storage(read, write)]
@> fn transfer_ownership(new_owner: Identity) {
@>     transfer_ownership(new_owner);
}
```

IMPACT

This may result in ownership being permanently untransferable.

RECOMMENDED MITIGATION

Either disambiguate the call to explicitly reference the imported function:

```
fn transfer_ownership(new_owner: Identity) {
-     transfer_ownership(new_owner);
+     ownership::transfer_ownership(new_owner);
}
```

Or rename the contract function to avoid shadowing:

```
- fn transfer_ownership(new_owner: Identity) {
+ fn transfer_owner(new_owner: Identity) {
    transfer_ownership(new_owner);
}
```

L-01 `get_price` Uses Pyth Price Without Validating Confidence Interval

Severity **Low** Status ● Resolved

DESCRIPTION

Within the `get_price` function, when Stork is either not configured or its price is stale, the contract falls back to fetching a price from Pyth. After fetching, the implementation only checks staleness via:

```
if current_time <= pyth_price.publish_time + TIMEOUT {
    _set_last_good_price(last_good_price, pyth_price.price, pyth_price.publish_time);
    return pyth_price.price;
}
```

However, Pyth explicitly recommends validating the confidence interval alongside the reported price. The confidence interval indicates the uncertainty of the price, particularly under volatile or illiquid market conditions. Relying solely on the publish time while ignoring confidence may lead to using unreliable or manipulated prices.

Best practice when integrating with Pyth feeds is to reject prices with excessively wide confidence intervals, as these may reflect temporary dislocations, low liquidity, or abnormal market events.

IMPACT

The protocol may accept Pyth prices that are recent but inaccurate or untrustworthy.

RECOMMENDED MITIGATION

Incorporate a confidence interval check when validating Pyth prices similar to Fluid

```
fn is_pyth_price_stale_or_outside_confidence(pyth_price: Price, current_time: u64) -> bool
{
    let confidence_threshold = pyth_price.price / 25;
    return current_time > pyth_price.publish_time + TIMEOUT || pyth_price.confidence >
confidence_threshold;
}
```

I-01 Missing Zero Address Validation in `transfer_ownership`

Severity **Informational** Status ● Resolved

DESCRIPTION

The `transfer_ownership` function currently forwards ownership transfer calls without checking whether the `new_owner` is a valid non-zero address. This means ownership can be assigned to the zero address, effectively renouncing ownership of the contract.

While renouncing ownership may be intentional in some cases, the `sway_libs::ownership` library already provides a dedicated `renounce_ownership` function for this purpose. Relying on `transfer_ownership` to achieve the same result is ambiguous and may create confusion.

I-02 `get_price` Always Prioritizes Stork Oracle, Ignoring Potentially Fresher Data

Severity Informational Status ● Resolved

DESCRIPTION

The `get_price` function implements a prioritization order when fetching asset prices. Currently, if Stork is configured and its price is not considered stale (`current_time <= stork_timestamp + TIMEOUT`), the function immediately returns the Stork price and does not evaluate data from Pyth or Redstone.

```
if current_time <= stork_timestamp + TIMEOUT {  
    _set_last_good_price(last_good_price, stork_price, stork_timestamp);  
    return stork_price; // Early exit  
}
```

This logic means that as long as Stork provides a price within the staleness window, it is treated as authoritative, regardless of whether Pyth or Redstone have fresher timestamps or more reliable/confident data.

While the comments in the contract suggest that Stork is intended to be the exclusive oracle for stFuel, the current implementation applies this preference broadly. This undermines the redundancy benefits of integrating multiple oracle providers.

IMPACT

As a result of this prioritization, `get_price` can potentially return less up-to-date prices and also introduces a single-point-of-failure risk

RECOMMENDED MITIGATION

Stork can be prioritized for stFuel or other designated assets; otherwise, refactor `get_price` to consider all configured oracle prices

APPENDIX A

Vulnerability checklist

Vulnerability classes checked during this review, as listed in the original report.

✓ Access Management	✓ Compiler version not fixed
✓ Arbitrary write to storage	✓ Address hardcoded
✓ Centralization of control	✓ Divide before multiply
✓ Ether theft	✓ Integer overflow/underflow
✓ Improper or missing events	✓ ERC compliance
✓ Logical issues and flaws	✓ Dangerous strict equalities
✓ Arithmetic Computations Correctness	✓ Tautology or contradiction
✓ Race conditions/front running	✓ Missing Zero Address Validation
✓ SWC Registry	✓ Private modifier
✓ Re-entrancy	✓ Revert/require functions
✓ Timestamp Dependence	✓ Multiple Sends
✓ Exception Disorder	✓ Using suicide
✓ Gasless Send	✓ Using delegatecall
✓ Use of tx.origin	✓ Upgradeable safety
✓ Malicious libraries	✓ Using throw

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS