



SMART CONTRACT SECURITY AUDIT

Moor

Token Airdrop

CLIENT

Rig Labs

PLATFORM

Fuel

LANGUAGE

Sway

REVIEW

November 2025

DOCUMENT

Document information

Project	Moor, Token Airdrop
Client	Rig Labs
Platform	Fuel
Language	Sway
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Comprehensive manual code review, static analysis, threat-modeling, and best-practice compliance verification
Timeframe	November 2025
Status at issue	Under review

Document history

VERSION	DATE	CHANGE
1.0	November 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 Incorrect Token Recipient in claim Function	8
	H-02 Incorrect Deadline Validation Using Mismatched Timestamp Formats	9
	L-01 Missing total_allocation Enforcement For Token Distribution in claim Function	9
	Disclaimer	11

Executive summary

About Moor

Moor is a token from Rig Labs on Fuel. The airdrop contract lets users claim MOOR allocations from a Merkle tree before a deadline, after which the owner can claw back unclaimed tokens.

Summary of the review

Safe Edges reviewed the Moor token airdrop contract on Fuel, written in Sway, in November 2025. Three issues were identified: two High and one Low. The High issues were claims that sent tokens to the caller instead of the named recipient, and deadline checks that compared a UNIX timestamp with Fuel's TAI64 block timestamp. The report was issued while remediation was under review, so no resolution status was recorded.

Findings by severity



Findings by status

STATUS	ISSUES
● Not recorded	3

Scope

Repository	Not recorded in the original report
------------	-------------------------------------

Commit	Not recorded in the original report
--------	-------------------------------------

Platform	Fuel
----------	------

Language	Sway
----------	------

IN-SCOPE FILES

```
Airdrop contract
MoorToken
```

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: Comprehensive manual code review, static analysis, threat-modeling, and best-practice compliance verification.

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	Incorrect Token Recipient in claim Function	High	● Not recorded
H-02	Incorrect Deadline Validation Using Mismatched Timestamp Formats	High	● Not recorded
L-01	Missing total_allocation Enforcement For Token Distribution in claim Function	Low	● Not recorded

Detailed findings

H-01 Incorrect Token Recipient in claim Function

Severity **High** Status ● Not recorded

DESCRIPTION

Within the claim function, the contract transfers the airdropped tokens to the caller rather than the specified recipient. This contradicts both the intended behavior of the function and the accompanying NatSpec comments, which indicate that the tokens should be sent to the designated recipient. As implemented, the recipient parameter has no effect on the actual transfer.

```
#[storage(read, write)]
fn claim(amount: u64, recipient: Identity, tree_index: u64, proof: Vec<b256>) {
    // Check if paused
    require(!storage.is_paused.read(), StateError::ContractPaused);
    //existing code
    // Transfer MOOR tokens to recipient
    @>> transfer(msg_sender_ident, storage.moor_asset.read(), amount);
```

IMPACT

Airdropped tokens are delivered to the caller (msg_sender) instead of the intended recipient, rendering the recipient argument functionally meaningless and potentially breaking workflows that rely on delegated claiming or custodial recipients.

RECOMMENDED MITIGATION

Consider adding these changes:

```
/// Errors related to input validation
pub enum InputError {
    /// The amount must be greater than zero
    InvalidAmount: (),
    /// The tree index is out of bounds
    InvalidIndex: (),
    + InvalidRecipient: (),
}
// Mark as claimed
storage.claims.insert(tree_index, true);
// Transfer MOOR tokens to recipient
- transfer(msg_sender_ident, storage.moor_asset.read(), amount);
+ require(recipient != Identity::Address(Address::zero()), InputError::InvalidRecipient);
+ transfer(recipient, storage.moor_asset.read(), amount);
```

H-02 Incorrect Deadline Validation Using Mismatched Timestamp Formats

Severity **High** Status ● Not recorded

DESCRIPTION

The deadline checks in both claim and clawback compare a UNIX timestamp against Fuels native block timestamp, `std::block::timestamp()`, which is expressed in TAI64 format. Because UNIX and TAI64 timestamps are not directly comparable, this check is inherently incorrect and results in flawed deadline validation.

```
#[storage(read, write)]
fn claim(amount: u64, recipient: Identity, tree_index: u64, proof: Vec<b256>) {
    // Check if paused
    require(!storage.is_paused.read(), StateError::ContractPaused);
    // Check if airdrop has expired
    @>> require(std::block::timestamp() <= storage.end_time.read(), TimingError::AirdropExpired);
#[storage(read)]
fn clawback(amount: u64, recipient: Identity) {
    // Only owner can clawback
    _only_owner();
    // Can only clawback after expiry
    @>> require(std::block::timestamp() > storage.end_time.read(), TimingError::AirdropNotExpired);
```

IMPACT

Airdrop expiry enforcement may fail entirely or behave unexpectedly, allowing claims after the intended deadline or preventing valid claims before it.

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.

L-01 Missing total_allocation Enforcement For Token Distribution in claim Function

Severity **Low** Status ● Not recorded

DESCRIPTION

The contract sets a `total_allocation` value during initialization to define the total number of tokens intended for airdrop distribution. However, this value is never enforced during the lifecycle of the contract. The claim function does not track cumulative claimed amounts nor validate that a new amount claim keeps the total distributed tokens within the declared allocation.

As a result, the contracts only safeguard is whether it physically holds enough tokens. If the contract is funded above `total_allocation`, and if the Merkle tree encodes leaves whose aggregate amount exceeds it, users can successfully claim amounts beyond the intended cap

IMPACT

total_allocation provides no real protection or enforcement

RECOMMENDED MITIGATION

Consider enforcing the declared total_allocation by introducing a total_claimed counter and rejecting any claim that would cause the cumulative distributed amount to exceed the allocation.

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