



SMART CONTRACT SECURITY AUDIT

Hyperlane

Fuel Integration

CLIENT

Hyperlane

PLATFORM

Fuel

LANGUAGE

Sway

REVIEW

March 2025

DOCUMENT

Document information

Project	Hyperlane, Fuel Integration
Client	Hyperlane
Platform	Fuel
Language	Sway
Repository	fuel-infrastructure/fuel-hyperlane-integration
Commit	941effc8bb46e3704cc4f947d81ef6ccdd5e4183
Methods	Manual Analysis, Functional Testing, Automated Testing
First review	1 to 10 March 2025
Updated code received	10 March 2025
Second review	10 March 2025

Document history

VERSION	DATE	CHANGE
1.0	10 March 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
	L-01 CEI Pattern Violation in c <code>laim</code> Function	8
	L-02 Deprecated Use of <code>ZERO_B256</code>	9
	L-03 Incorrect Handling of <code>msg_amount()</code> in <code>Mailbox::dispatch</code>	10
A	Appendix: vulnerability checklist	11
	Disclaimer	12

Executive summary

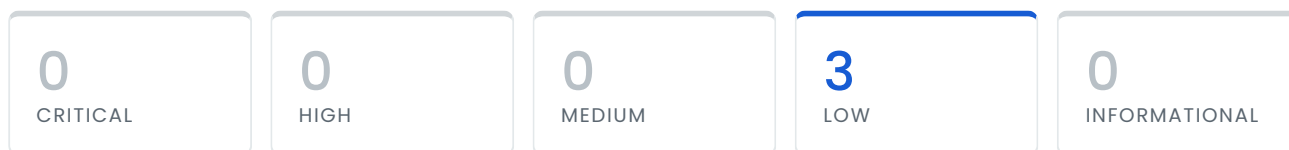
About Hyperlane

Hyperlane is a permissionless interoperability protocol for cross-chain communication. It enables message passing and asset transfers across different chains without relying on centralized intermediaries or requiring any permissions.

Summary of the review

Safe Edges reviewed Hyperlane's Fuel integration, the Sway contracts that bring Hyperlane's cross-chain messaging to the Fuel network. The review ran from 1 to 10 March 2025 using manual analysis, functional testing and automated testing. Three Low severity issues were identified and no Critical, High or Medium issues were found. The Hyperlane team addressed all three, and the fixes were confirmed in a review of the updated code on 10 March 2025.

Findings by severity



Findings by status

STATUS	ISSUES
● Resolved	3

Scope

Repository	fuel-infrastructure/fuel-hyperlane-integration
Commit	941effc8bb46e3704cc4f947d81ef6ccdd5e4183
Platform	Fuel
Language	Sway

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 Analysis, Functional Testing, Automated Testing.

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
L-01	CEI Pattern Violation in <code>claim</code> Function	Low	● Resolved
L-02	Deprecated Use of <code>ZERO_B256</code>	Low	● Resolved
L-03	Incorrect Handling of <code>msg_amount()</code> in <code>Mailbox::dispatch</code>	Low	● Resolved

Detailed findings

L-01 CEI Pattern Violation in `claim` Function

Severity Low Status Resolved

DESCRIPTION

The `claim` function in the contract does not follow the Checks-Effects-Interactions (CEI) pattern, which is a best practice for secure smart contract design. The function performs an external call (`transfer`) before updating storage, which could lead to minor inconsistencies in contract state. However, since the function is restricted to only the owner, the risk of reentrancy is none.

AFFECTED CODE

Function: `claim`

```
fn claim(asset: Option<AssetId>) {
    only_owner();
    let beneficiary = storage.beneficiary.read();
    let stored_asset = storage.asset_id.read();
    let asset = asset.unwrap_or(stored_asset);
    let balance = this_balance(asset);

    transfer(beneficiary, asset, balance); // @audit Transfer before storage update

    if stored_asset == asset {
        storage.contract_balance.write(0);
    }

    log(ClaimEvent {
        beneficiary,
        amount: balance,
    });
}
```

IMPACT

- **Minimal Security Risk:** The function is restricted to the owner, so reentrancy or malicious behavior is unlikely. However, following best practices ensures long-term security.

RECOMMENDATION

Follow the CEI pattern by updating storage before transferring funds.

RECOMMENDED CODE

```
fn claim(asset: Option<AssetId>) {
    only_owner();
    let beneficiary = storage.beneficiary.read();
    let stored_asset = storage.asset_id.read();
    let asset = asset.unwrap_or(stored_asset);
    let balance = this_balance(asset);

    // Update storage before external call
    if stored_asset == asset {
        storage.contract_balance.write(0);
    }

    // Perform the external call after storage update
    transfer(beneficiary, asset, balance);

    log(ClaimEvent {
        beneficiary,
        amount: balance,
    });
}
```

L-02 Deprecated Use of ZERO_B256

Severity Low Status Resolved

DESCRIPTION

The contract makes use of `constants::ZERO_B256`, which is now deprecated according to the updated Sway language standards. Using deprecated constants may lead to future compatibility issues as the language evolves. The recommended approach is to use `b256::zero()` instead, which aligns with the latest standards.

AFFECTED CODE

```
let zero_hash = constants::ZERO_B256; // @audit Deprecated usage
```

IMPACT

- **Future Compatibility Issues:** The use of `ZERO_B256` may result in warnings or errors in future versions of the Sway compiler.
- **Code Maintainability:** Keeping the contract up to date with the latest Sway standards ensures long-term maintainability and easier debugging.

RECOMMENDATION

Replace all occurrences of `constants::ZERO_B256` with `b256::zero()` to comply with updated Sway language standards.

RECOMMENDED CODE

```
let zero_hash = b256::zero(); // Updated as per latest Sway standards
```

L-03 Incorrect Handling of `msg_amount()` in `Mailbox::dispatch`

Severity

Low

Status ● Resolved

DESCRIPTION

In the `Mailbox::dispatch` function (line 271), there is a check:

```
if (msg_amount() < required_value) { required_value = msg_amount(); }
```

This logic sets `required_value` to `msg_amount()` if the sent amount is less than the required amount. However, this can lead to unintended reverts during external contract interactions, as the function will proceed with a lower value than required.

To prevent unnecessary contract execution, it is recommended to revert immediately if `msg_amount()` is insufficient, rather than adjusting `required_value` downward.

IMPACT

- **Unnecessary External Calls:** The function proceeds with an incorrect `required_value`, which ultimately causes reverts later in the execution.
- **Gas Inefficiency:** Wasted computation and failed transactions could increase user costs.

RECOMMENDATION

Instead of modifying `required_value`, the function should revert immediately when `msg_amount()` is insufficient.

RECOMMENDED CODE

```
if (msg_amount() < required_value) { revert("Insufficient funds sent for operation") }
```

APPENDIX A

Vulnerability checklist

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

✓ Arbitrary write to storage	✓ Upgradeable safety
✓ Centralization of control	✓ Using inline assembly
✓ Ether theft	✓ Style guide violation
✓ Improper or missing events	✓ Parallel Execution safety
✓ Logical issues and flaws	✓ UTXO Model Verification
✓ Arithmetic Computations Correctness	✓ FuelVM Opcodes
✓ Race conditions/front running	✓ Cross-Chain Interactions
✓ Re-entrancy	✓ Modular Design
✓ Malicious libraries	✓ Access Control Vulnerabilities
✓ Address hardcoded	✓ Denial of Service (DoS)
✓ Divide before multiply	✓ Oracle Manipulation
✓ Integer overflow/underflow	✓ Signature Replay Attacks
✓ ERC's conformance	✓ Improper Handling of External Calls
✓ Missing Zero Address Validation	✓ Proxy Storage Collision
✓ Revert/require functions	✓ Use of Deprecated Functions

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