



SMART CONTRACT SECURITY AUDIT

Mira

v1 Core and Periphery

CLIENT

Mira

PLATFORM

Fuel

LANGUAGE

Sway

REVIEW

September 2024

DOCUMENT

Document information

Project	Mira, v1 Core and Periphery
Client	Mira
Platform	Fuel
Language	Sway
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Audit period	26 August to 11 September 2024

Document history

VERSION	DATE	CHANGE
1.0	11 September 2024	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 Deadline Depends on Block Number Instead of Timestamp	8
	L-02 Lack of Revert in get_y Function Poses Convergence Risk	9
	L-03 Strict Deadline Check Prevents Exact Block Height Matching	10
	L-04 Single-Step Ownership Transfer in AMM Contract	10
	I-01 Prevalence of Magic Numbers in Codebase	11
	I-02 Potential Infinite Loop in Pool Functions	12
07	Audit conclusion	14
	Disclaimer	15

Executive summary

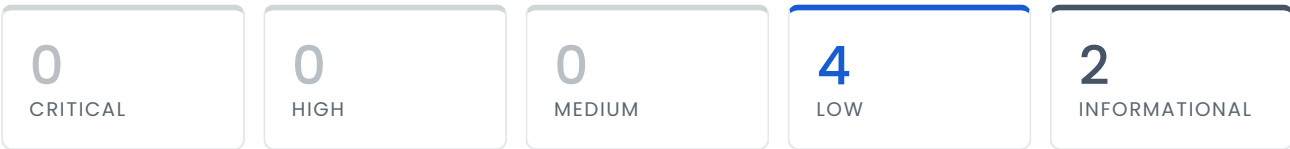
About Mira

Mira is an automated market maker (AMM) on the Fuel network. This audit covered the Mira v1 core and periphery contracts.

Summary of the review

Safe Edges reviewed Mira v1 core and periphery between 26 August and 11 September 2024. Six issues were identified: four Low and two Informational, with no Critical, High or Medium issues. At the time of the report one issue was resolved, three were acknowledged or accepted by the Mira team, and two remained open.

Findings by severity



Findings by status

STATUS	ISSUES
● Acknowledged	3
● Open	2
● Resolved	1

Scope

Repository	Not recorded in the original report
Commit	Not recorded in the original report
Platform	Fuel
Language	Sway

FOCUS AREAS

- Mira v1 Core & Periphery

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: Not recorded in the original report.

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	Deadline Depends on Block Number Instead of Timestamp	Low	● Acknowledged
L-02	Lack of Revert in get_y Function Poses Convergence Risk	Low	● Open
L-03	Strict Deadline Check Prevents Exact Block Height Matching	Low	● Resolved
L-04	Single-Step Ownership Transfer in AMM Contract	Low	● Acknowledged
I-01	Prevalence of Magic Numbers in Codebase	Informational	● Open
I-02	Potential Infinite Loop in Pool Functions	Informational	● Acknowledged

Detailed findings

L-01 Deadline Depends on Block Number Instead of Timestamp

Severity **Low** Status ● Acknowledged

DESCRIPTION

The protocol uses block height (`height()`) to enforce deadline logic. However, block height is not a consistent measure of time, which may lead to unexpected execution delays.

CODE EXAMPLE

```
pub fn check_deadline(deadline: u32) {  
    require(deadline >= height(), "Deadline passed");  
}
```

ISSUE

Due to inconsistent block production times, a transaction may succeed later than expected, potentially causing:

- Unexpected delays in transaction execution
- User confusion about transaction timing
- Potential front-running opportunities

RECOMMENDATION

Use timestamps instead of block height for more accurate time-based checks:

```
let start = timestamp() + MINIMUM_DELAY;  
let end = timestamp() + MAXIMUM_DELAY;  
require(start <= time && time <= end,  
    TransactionError::TimestampNotInRange((start, end, time)));
```

REMEDIATION

Risk Accepted — Mira team has accepted the limitations and decided to maintain current implementation.

L-02 Lack of Revert in get_y Function Poses Convergence Risk

Severity Low Status ● Open

DESCRIPTION

The `get_y` function attempts to calculate a value using iterations but does not revert even if convergence fails after 255 attempts.

ISSUE

Returning a potentially invalid y value could lead to:

- Inaccurate pricing calculations
- Potential user losses
- Protocol instability

CURRENT IMPLEMENTATION

```
fn get_y(x_0: u256, xy: u256, y: u256) -> u256 {
    let mut i = 0;
    while i < 255 {
        // Attempt convergence
        i += 1;
    }
    y // returns potentially incorrect value
}
```

RECOMMENDATION

Implement proper error handling when convergence fails:

```
fn get_y(x_0: u256, xy: u256, y: u256) -> u256 {
    let mut i = 0;
    while i < 255 {
        // Attempt convergence logic
        if (convergence_achieved) {
            return y;
        }
        i += 1;
    }
    assert(false, "Y convergence failed after 255 iterations");
    0 // Unreachable, but required
}
```

REMEDIATION

Unresolved — No reversion logic has been added to handle convergence failures.

L-03 Strict Deadline Check Prevents Exact Block Height Matching

Severity Low Status Resolved

DESCRIPTION

The deadline check only allows execution before the deadline, not at the deadline block height, which may confuse users expecting inclusive deadline behavior.

FLAWED LOGIC

```
require(deadline > height(), "Deadline passed");
```

ISSUE

- Transaction fails if deadline equals current block height
- Inconsistent with user expectations
- Reduces usability

RECOMMENDATION

Use inclusive deadline checking:

```
require(deadline >= height(), "Deadline passed");
```

REMEDIATION

Resolved — Code has been updated to use inclusive deadline checking.

L-04 Single-Step Ownership Transfer in AMM Contract

Severity Low Status Acknowledged

DESCRIPTION

The contract allows direct ownership transfers without confirmation from the new owner, creating risk of accidental or malicious transfers.

RISK

Could lead to:

- Accidental transfer to incorrect address
- Loss of contract control
- Malicious ownership hijacking

CURRENT PATTERN

```
fn transfer_ownership(new_owner: Identity) {  
    if _owner() == State::Uninitialized {  
        initialize_ownership(new_owner);  
    } else {  
        transfer_ownership(new_owner);  
    }  
}
```

RECOMMENDATION

Implement a secure 2-step ownership transfer process:

```
pub struct Contract {  
    owner: Identity,  
    pending_owner: Option<Identity>,  
}  
  
fn transfer_ownership(new_owner: Identity) {  
    require(msg.sender() == self.owner, "Not owner");  
    self.pending_owner = Some(new_owner);  
}  
  
fn accept_ownership() {  
    require(Some(msg.sender()) == self.pending_owner, "Not pending owner");  
    self.owner = msg.sender();  
    self.pending_owner = None;  
}
```

REMEDIATION

Acknowledged — Team noted that 2-step ownership transfer is not yet supported in Fuel standard library.

I-01 Prevalence of Magic Numbers in Codebase

Severity Informational Status ● Open

DESCRIPTION

Hardcoded values like 255, `0x3u256`, and division by 5 appear throughout the codebase without clear documentation or named constants.

EXAMPLES

```
amounts_in.get(amounts_in.len() - i - 2)  
0x3u256 * x_0  
for i in range(255)  
volatile_fee <= LP_FEE_VOLATILE / 5
```

ISSUE

Magic numbers reduce:

- Code readability
- Maintainability
- Developer understanding
- Audit efficiency

RECOMMENDATION

Replace magic numbers with well-documented constants:

```
const MAX_ITERATIONS: u256 = 255;
const CALCULATION_MULTIPLIER: u256 = 0x3u256;
const VOLATILE_FEE_DIVISOR: u256 = 5;

// Usage
for i in range(MAX_ITERATIONS)
    volatile_fee <= LP_FEE_VOLATILE / VOLATILE_FEE_DIVISOR
```

REMEDIATION

Unresolved — Magic numbers remain in the codebase without documentation.

I-02 Potential Infinite Loop in Pool Functions

Severity Informational Status ● Acknowledged

DESCRIPTION

Functions like `get_amounts_out` and `get_amounts_in` loop through the pools array without explicit iteration limits, potentially causing gas issues.

EXAMPLE

```
while (i < pools.len()) {
    // processing logic
    i += 1;
}
```

ISSUE

Large pool arrays could cause:

- Excessive gas consumption
- Transaction failures
- Poor user experience

RECOMMENDATION

Implement iteration limits with proper error handling:

```
const MAX_POOL_ITERATIONS: u64 = 256;

fn safe_pool_iteration(pools: &[Pool]) {
    let mut iteration_count: u64 = 0;
    let mut i = 0;
    while (i < pools.len() && iteration_count < MAX_POOL_ITERATIONS) {
        // processing logic
        i += 1;
        iteration_count += 1;
    }
    if iteration_count == MAX_POOL_ITERATIONS {
        revert("Maximum iteration limit reached");
    }
}
```

REMEDIATION

Acknowledged — Team accepts current implementation, trusting users to manage pool array sizes appropriately.

Audit conclusion

Overall security rating: Good.

Key takeaways

- No Critical or High-severity vulnerabilities were identified
- Most issues are related to code quality and best practices
- 1 out of 6 issues has been fully resolved
- Protocol demonstrates good security fundamentals

Recommendations for future development

1. Time Management: Consider migrating to timestamp-based deadlines for better user experience
2. Error Handling: Implement proper convergence failure handling in mathematical functions
3. Code Quality: Replace magic numbers with named constants for better maintainability
4. Gas Optimization: Consider implementing iteration limits for large array operations

While most identified issues are low-severity or informational in nature, addressing them can significantly improve long-term maintainability, user experience, and trust in the Mira Protocol.

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