Mira Protocol - Security Audit Report

Audit Conducted By: Safe Edges
Website: https://safeedges.in
Audit Scope: Mira v1 Core & Periphery
Audit Period: August 26, 2024 – September 11, 2024

Summary of Findings

ID	Title	Severity	Status
1	Deadline Depends on Block Number Instead of Timestamp	Low	Risk Accepted
2	Lack of Revert in `get_y` Function Poses Convergence Risk	Low	Unresolved
3	Strict Deadline Check Prevents Exact Block Height Matching	Low	Resolved
4	Single-Step Ownership Transfer in AMM Contract	Low	Acknowledged
5	Prevalence of Magic Numbers in Codebase	Informational	Unresolved
6	Potential Infinite Loop in Pool Functions	Informational	Acknowledged

Issue Distribution

Severity Level	Count	Status Overview
Low	4	1 Resolved, 1 Risk Accepted, 2 Outstanding
Informational	2	2 Acknowledged
Total	**6**	**1 Fully Resolved**

1. Deadline Depends on Block Number Instead of Timestamp

Severity: Low | **Status:** Risk Accepted

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.

---

2. Lack of Revert in `get_y` Function Poses Convergence Risk

Severity: Low | **Status:** Unresolved

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.

---

3. 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.

---

4. 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.

---

5. Prevalence of Magic Numbers in Codebase

Severity: Informational | **Status:** Unresolved

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.

---

6. 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

Security Assessment Summary

Overall Security Rating	GOOD
Critical Issues	0
High Issues	0
Medium Issues	0
Low Issues	4
Informational	2

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

---

Contact Information

Safe Edges - Trusted Experts in Smart Contract Security

This audit was conducted by Safe Edges, a premier blockchain security firm specializing in comprehensive smart contract audits and security assessments.

For follow-up consultations or additional security assessments:
https://safeedges.in

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.

Report Prepared By: Safe Edges Security Team
Date: September 11, 2024