
# Mira

## Security Assessment

October 10th, 2024 — Prepared by OtterSec

Renato Eugenio Maria Marziano renato@osec.io

James Wang james.wang@osec.io

# Table of Contents

# 01 — Executive Summary

## Overview

Mira engaged OtterSec to assess the `mira-amm` program. This assessment was conducted between October 2nd and October 7th, 2024. For more information on our auditing methodology, refer to Appendix B.

## Key Findings

We produced 4 findings throughout this audit engagement.

In particular, we identified a vulnerability concerning the utilization of inappropriate default values for nonexistent assets, which may result in incorrect configurations in the pool when assets are created later (OS-MRA-ADV-00). Additionally, we highlighted a possible underestimation of the required input amount for swaps in the non-stable branch due to incorrect rounding (OS-MRA-ADV-01).

We also made a recommendation to add a check before transfer operations to ensure that the amount to be transferred is greater than zero (OS-MRA-SUG-00) and advised implementing additional checks and adjustments to ensure accurate swap amount calculations (OS-MRA-SUG-01).

# 02 — Scope

The source code was delivered to us in a Git repository at https://github.com/mira-amm. This audit was performed against a2c5788 and 0ebb288.

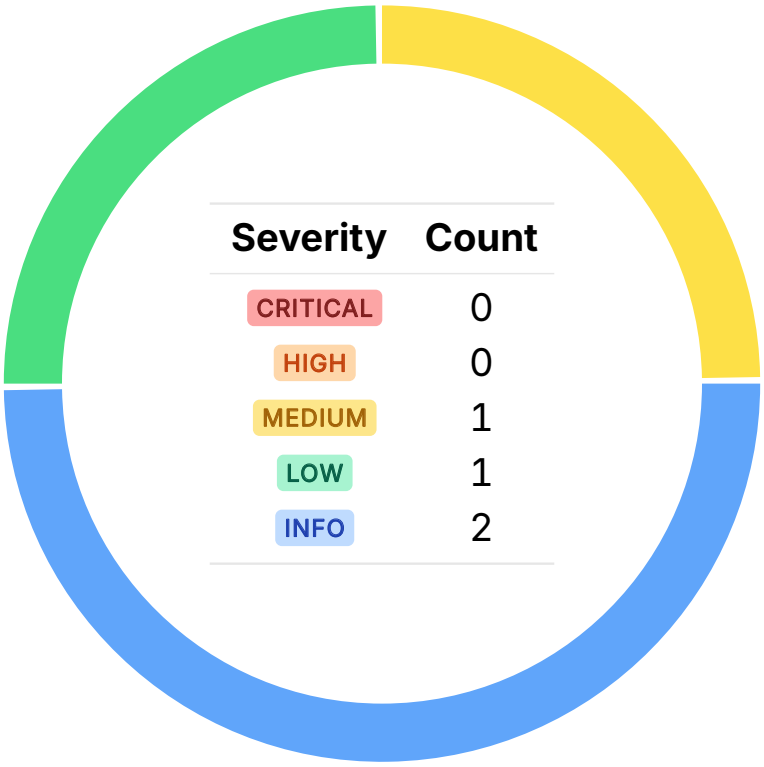
**A brief description of the program is as follows:**

| Name | Description |
| --- | --- |
| mira-amm | Implementation of an automated market maker, allowing for decentralized token swaps and liquidity provisioning on the Fuel chain. |

# 03 — Findings

Overall, we reported 4 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



| Severity | Count |
| --- | --- |
| CRITICAL | 0 |
| HIGH | 0 |
| MEDIUM | 1 |
| LOW | 1 |
| INFO | 2 |

# 04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

| ID | Severity | Status | Description |
|---|---|---|---|
| OS-MRA-ADV-00 | MEDIUM | RESOLVED | `get_symbol_and_decimals` utilizes inappropriate default values for nonexistent assets, which may result in incorrect configurations in the pool when assets are created later. |
| OS-MRA-ADV-01 | LOW | RESOLVED | `get_amount_in` may underestimate the required input amount for swaps in the non-stable branch due to incorrect rounding. |

## Inappropriate Default Values for Nonexistent Assets MEDIUM OS-MRA-ADV-00

### Description

`get_symbol_and_decimals` utilizes inappropriate default values when querying the symbol and decimals of a given asset via the `src_20` interface. The function attempts to retrieve the symbol and decimals for an asset utilizing the provided `asset_id`. If either the `symbol` or `decimals` method returns `None`, the function falls back to default values of `UNKNOWN` and zero for the `symbol` and `decimals`, respectively.

*>_ libraries/utils/src/src20_utils.sw* RUST

```rust
pub fn get_symbol_and_decimals(contract_id: ContractId, asset_id: AssetId) -> (String, u8) {
    let src_20 = abi(SRC20, contract_id.into());
    let symbol = src_20.symbol(asset_id).unwrap_or(String::from_ascii_str("UNKNOWN"));
    let decimals = src_20.decimals(asset_id).unwrap_or(0);
    (symbol, decimals)
}
```

If a pool utilizes the default values for an asset that does not exist, it will result in mismatches when the asset is created later. Specifically, if the actual asset has a different decimal count than zero, it will lead to incorrect calculations.

### Remediation

Restrict the creation of pools for nonexistent assets.

### Patch

Resolved in 0ebb288.

## Incorrect Rounding of Input Amount LOW OS-MRA-ADV-01

### Description

`get_amount_in` in `pool_math` fails to round up when calculating the input amount for a swap in non-stable pools. If the input amount is underestimated due to rounding down, the user may provide less than what is required to complete the swap. In such cases, the swap may fail.

### Remediation

Round up the result to ensure the input amount is overestimated rather than underestimated. This way, users provide a slightly higher amount of tokens, ensuring the swap succeeds without falling short of the required input.

### Patch

Resolved in 14ac97e.

# 05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

| ID | Description |
| --- | --- |
| OS-MRA-SUG-00 | `burn` may attempt to transfer zero amounts of assets, resulting in the transaction being reverted. |
| OS-MRA-SUG-01 | The current implementation of Newton’s method may not converge in certain edge cases, necessitating additional final checks and adjustments to ensure accurate swap amount calculations. |

## Possibility of Reverting a Zero Transfer OS-MRA-SUG-00

### Description

In `burn`, when calculating the amounts of asset zero and asset one to be transferred back to the liquidity provider after burning liquidity provider tokens, there may be situations where the calculated amounts result in zero due to rounding. This will result in the subsequent transfer operation reverting due to a zero-transfer scenario.

*>_ mira-v1-core/contracts/mira_amm_contract/src/main.sw* RUST

```rust
fn burn(pool_id: PoolId, to: Identity) -> (u64, u64) {
    [...]
    let asset_0_out = proportional_value(burned_liquidity.amount,
        ↪  pool.reserve_0,total_liquidity);
    let asset_1_out = proportional_value(burned_liquidity.amount,
        ↪  pool.reserve_1,total_liquidity);
    transfer(to, pool_id.0, asset_0_out);
    transfer(to, pool_id.1, asset_1_out);
    update_reserves(pool, 0, 0, asset_0_out, asset_1_out);
    [...]
}
```

### Remediation

Add a check before the transfer operations to ensure that the amount to be transferred is greater than zero.

## Ensuring Accurate Swap Calculations OS-MRA-SUG-01

### Description

Within `pool_math::get_y`, it is advisable to implement final checks and adjustments to the output of Newton's method when calculating swap amounts. This ensures that the result is valid and has converged to a meaningful solution, thereby preventing potential edge cases where Newton's method may fail to converge.

### Remediation

Implement the above-mentioned suggestion.

# A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

**CRITICAL**

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

**HIGH**

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

**MEDIUM**

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

**LOW**

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

**INFO**

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

# B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.