Math Masters Audit Report

Version 1.0

Lokapal

February 16, 2025

Math Masters Audit Report

Lokapal

February 16, 2025

Prepared by:

- Lokapal

Lead Auditor:

- Ricardo Pintos

Table of Contents

    * [L-1] Incorrect memory access in `MathMasters::mulWad` and `MathMasters::mulWadUp` leading to a blank message revert
  - Informational
    * [I-1] Solidity version `0.8.3` does not support custom errors
    * [I-2] `MathMasters::mulWad` and `MathMasters::mulWadUp` have the wrong function selector for `MathMasters__MulWadFailed()`

Protocol Summary

This protocol is an arithmetic library with operations for fixed-point numbers.

Disclaimer

The LOKAPAL team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

- Commit Hash:

```
c7643faa1a188a51b2167b68250816f90a9668c6
```

- In Scope:

```
# MathMasters.sol
```

Executive Summary

This security review was conducted as part of Cyfrin Updraft's Formal Verification course.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Info	2
Total	5

Findings

High

[H-1] Incorrect rounding in `MathMasters::mulWadUp` leading to miscalculations

Description: The `mulWadUp` function has an extra line that checks the inputs and adds a 1 to the calculation if the conditional passes:

```
function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256 z) {
    assembly {
        if mul(y, gt(x, div(not(0), y))) {
            mstore(0x40, 0xbac65e5b)
            revert(0x1c, 0x04)
        }
~~>         if iszero(sub(div(add(z, x), y), 1)) { x := add(x, 1) }
        z := add(iszero(iszero(mod(mul(x, y), WAD))), div(mul(x, y), WAD))
    }
}
```

Impact: This line is not needed to perform the proper rounding up correction. But, in corner cases, this conditional adds a 1 unnecessarily, disrupting the final input and returning a wrong rounded up number.

Proof of Concept:

1. HALMOS: You can use this `Halmos` test to formally verify this error. Run `halmos --function check_testMulWadUpHalmos`

```
function check_testMulWadUpHalmos(uint256 x, uint256 y) public pure {
    if (x == 0 || y == 0 || y <= type(uint256).max / x) {
        uint256 result = MathMasters.mulWadUp(x, y);
        uint256 expected = x * y == 0 ? 0 : (x * y - 1) / 1e18 + 1;
        assert(result == expected);
    }
}
```

2. CERTORA: You can use the files in the `./certora/MulWadUp/` folder to run Certora to formally verify this error. Also, you can see this Certora Job Report.
3. UNIT TEST: You can run the `testMulWadUpUnitFromHalmos` and `testMulWadUpUnitFromCertora` tests in the `ProofsOfCode.t.sol` file to see examples of inputs that cause calculation errors.

Recommended mitigation: Consider removing the extra line to avoid errors:

```
function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256 z) {
    assembly {
        if mul(y, gt(x, div(not(0), y))) {
            mstore(0x40, 0xbac65e5b)
            revert(0x1c, 0x04)
        }
-           if iszero(sub(div(add(z, x), y), 1)) { x := add(x, 1) }
        z := add(iszero(iszero(mod(mul(x, y), WAD))), div(mul(x, y), WAD))
    }
}
```

[H-2] Incorrect value in `MathMasters::sqrt` leading to miscalculations

Description: The `sqrt` function performs the Babylonian method to find the square root of an input. There is an incorrect value in the specified line:

```
1  function sqrt(uint256 x) internal pure returns (uint256 z) {
2      assembly {
3          z := 181
4          let r := shl(7, lt(87112285931760246646623899502532662132735, x))
5          r := or(r, shl(6, lt(4722366482869645213695, shr(r, x))))
6          r := or(r, shl(5, lt(1099511627775, shr(r, x))))
7  ~~>     r := or(r, shl(4, lt(16777002, shr(r, x))))
8          z := shl(shr(1, r), z)
9
10     // FUNCTION CONTINUES...
11     }
12 }
```

Impact: In corner cases, this value will create a wrong output. This error can be found using fuzz tests only if the number of runs is extremely high, so the best approach is a formal verification tool.

Proof of Concept: Running the `sqrt` function directly with the Certora Prover will result in a Path Explosion issue. So, we use the upper halves of the `sqrt` function and the `Base_Test::solmateSqrt` function. They should return the same result when passing the same input.

1. CERTORA: You can use the files in the `./certora/Sqrt/` folder to run Certora to formally verify this error. Also, you can see this Certora Job Report.
2. UNIT TEST: You can run the `testSqrtFailedFromCertora` test in the `ProofsOfCode.t.sol` file to see an example of an input that causes calculation errors.

Recommended mitigation: Consider replacing the wrong value with the correct value:

```
1  function sqrt(uint256 x) internal pure returns (uint256 z) {
2      assembly {
3          z := 181
4          let r := shl(7, lt(87112285931760246646623899502532662132735, x))
5          r := or(r, shl(6, lt(4722366482869645213695, shr(r, x))))
6          r := or(r, shl(5, lt(1099511627775, shr(r, x))))
7  +       r := or(r, shl(4, lt(16777215, shr(r, x))))
8  -       r := or(r, shl(4, lt(16777002, shr(r, x))))
9          z := shl(shr(1, r), z)
10
```

```
11      // FUNCTION CONTINUES...
12      }
13  }
```

Low

[L-1] Incorrect memory access in `MathMasters::mulWad` and `MathMasters::mulWadUp` leading to a blank message revert

Description: The `mulWad` and `mulWadUp` functions use assembly to store a function selector in memory, which is later used for reverting. However, the `mstore` command places this selector at the Free Memory Pointer location, causing a mismatch between where the selector is stored and where the `revert` command retrieves it.

```
1  function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256 z) {
2      assembly {
3          if mul(y, gt(x, div(not(0), y))) {
4  ~~>         mstore(0x40, 0xbac65e5b)
5              revert(0x1c, 0x04)
6          }
7              if iszero(sub(div(add(z, x), y), 1)) { x := add(x, 1) }
8          z := add(iszero(iszero(mod(mul(x, y), WAD))), div(mul(x, y), WAD))
9      }
10 }
```

Recommended mitigation: If you still want to use assembly for this procedure, consider avoiding the Free Memory Pointer location and calculate the proper location used by the `revert` command.

```
1  function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256 z) {
2      assembly {
3          if mul(y, gt(x, div(not(0), y))) {
4  +           mstore(0x80, 0xbac65e5b)
5  +           revert(0x9c, 0x04)
6  -           mstore(0x40, 0xbac65e5b)
7  -           revert(0x1c, 0x04)
8          }
9              if iszero(sub(div(add(z, x), y), 1)) { x := add(x, 1) }
10         z := add(iszero(iszero(mod(mul(x, y), WAD))), div(mul(x, y), WAD))
11     }
12 }
```

Informational

[I-1] Solidity version `0.8.3` does not support custom errors

Description: The `MathMasters` library uses custom errors:

```
    error MathMasters__FactorialOverflow();
    error MathMasters__MulWadFailed();
    error MathMasters__DivWadFailed();
    error MathMasters__FullMulDivFailed();
```

Custom errors were introduced in version `0.8.4`, so the Solidity compiler won't recognize the error declaration.

Recommended mitigation: To avoid this issue, you should either:

- Update your `pragma` to `>=0.8.4` if possible.
- Use traditional require/revert statements with error messages instead of custom errors.

[I-2] `MathMasters::mulWad` and `MathMasters::mulWadUp` have the wrong function selector for `MathMasters__MulWadFailed()`

Description: The `mulWad` functions in `MathMasters` uses assembly to push on the stack the function selector of an error:

```
    if mul(y, gt(x, div(not(0), y))) {
        mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
        revert(0x1c, 0x04)
    }
```

The documentation states that the intended custom error to be displayed is `MathMasters__MulWadFailed()`. But the correct function selector for that error is `0xa56044f7`.

Recommended mitigation: Use the correct function selector for the intended custom error:

```
    if mul(y, gt(x, div(not(0), y))) {
+       mstore(0x40, 0xa56044f7) // `MathMasters__MulWadFailed()`.
-       mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
        revert(0x1c, 0x04)
    }
```