MathMasters Audit Report

Version 1.0

César Escribano

MathMasters Audit Report

César Escribano

January 31, 2024

Prepared by: César Escribano (@ceseshi)

Table of Contents

Protocol Summary

This is a security review of First Flight #8: Math Master, a public contest from CodeHawks.

This contract is a math library, optimized for gas efficiency.

Disclaimer

The auditor 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 auditor 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

Repository

https://github.com/Cyfrin/2024-01-math-master

Commit Hash

```
84c149baf09c1558d7ba3493c7c4e68d83e7b3aa
```

Scope

```
#-- MathMasters.sol
```

Roles

Executive Summary

This audit took place in January 2024, over 4 days, totalling 16 hours. The tools used were Visual Studio Code, Foundry and Halmos.

Issues found

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

Findings

High

[H-1] Incorrect input checking in mulWadUp may lead to calculation errors

Relevant GitHub Links

https://github.com/Cyfrin/2024-01-math-master/blob/84c149baf09c1558d7ba3493c7c4e68d83e7b3aa/src/MathMaster

Summary

The mulWadUp() function must check that the result of x * y will not overflow, but the condition is incorrect.

Vulnerability Details

The condition `if mul(y, gt(x, or(div(not(0), y), x)))` is incorrect, as it allows for x * y to overflow.

```
1  function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256
       z) {
2      /// @solidity memory-safe-assembly
3      assembly {
4          // Equivalent to `require(y == 0 || x <= type(uint256).max / y)
               `.
5  @>          if mul(y, gt(x, or(div(not(0), y), x))) {
6              mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
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 }
```

Impact

The function will return incorrect calculations, which may lead to loss of funds if used in monetary transactions.

Tools Used

Foundry, Manual review

Proof of Concept

This is a test that should revert, but it doesn't.

```
1  function testMulWadUpOverflow1() public {
2      uint256 y = 100;
3      uint256 x = (type(uint256).max / y) + 1;
4      uint256 r = MathMasters.mulWadUp(x, y);
5      console2.log("r", r);
6  }
```

Test passes, confirming that the returned value did overflow.

```
1  forge test --mt testMulWadOverflow
2  ...
3  [PASS] testMulWadUpOverflow1() (gas: 3775)
4  Logs:
5    r 1
```

Recommended Mitigation

Correct the condition.

```
function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256 z) {
    /// @solidity memory-safe-assembly
    assembly {
        // Equivalent to `require(y == 0 || x <= type(uint256).max / y)`.
-           if mul(y, gt(x, or(div(not(0), y), x))) {
+           if mul(y, gt(x, div(not(0), y))) {
            mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
            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))
    }
}
```

Test it:

```
function testMulWadUpRevert() public {
    uint256 y = 100;
    uint256 x = (type(uint256).max / y) + 1;
    vm.expectRevert(0xa56044f7);
    uint256 r = MathMasters.mulWadUp(x, y);
    console2.log("r", r);
}
```

Test passes, confirming that the function reverted correctly.

```
forge test --mt testMulWadUpRevert

...
[PASS] testMulWadUpRevert() (gas: 3275)
```

[H-2] Incorrect logic in mulWadUp may lead to calculation errors

Relevant GitHub Links

https://github.com/Cyfrin/2024-01-math-master/blob/84c149baf09c1558d7ba3493c7c4e68d83e7b3aa/src/MathMaster

Summary

The mulWadUp() function has a condition to check if `x / y == 1`, and in that case it adds 1 to x. This is unnecessary, and can lead to errors.

Vulnerability Details

The condition `if iszero(sub(div(add(z, x), y), 1))` checks if `x / y == 1`. This may have been added to adjust rounding upwards, but it is unnecessary and for big numbers may cause

errors and overflows.

```
function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256 z) {
    /// @solidity memory-safe-assembly
    assembly {
        // Equivalent to `require(y == 0 || x <= type(uint256).max / y)`.
        if mul(y, gt(x, div(not(0), y))) {
            mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
            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

The function will return incorrect calculations, which may lead to loss of funds if used in monetary transactions.

Tools Used

Foundry, Halmos, Manual review

Proof of Concept

This test should pass but will fail.

```
function testMulWadUpOverflow2() public {
    uint256 x2 = 14945383340880163210026968995183628808 9;
    uint256 y2 = 7970098193717564942745135657100143327 7;
    assertEq(MathMasters.mulWadUp(x2, y2), (x2 * y2 - 1) / 1e18 + 1);
}
```

This test will overflow.

```
function testMulWadUpOverflow3() public {
    uint256 y = MathMasters.sqrt(type(uint256).max);
    uint256 x = type(uint256).max / y + 1;
    uint256 r = MathMasters.mulWadUp(x, y);
    console2.log("r", r);
}
```

```
forge test --mt testMulWadUpOverflow2
...

[FAIL. Reason: assertion failed] testMulWadUpOverflow2() (gas: 15259)
Logs:
```

```
  Error: a == b not satisfied [uint]
        Left:
           119116172769565574971083803648853877292536938543395611848 17
       Right:
          11911617276956557497108380364885387729173992872402385535389
...

```bash
forge test --mt testMulWadUpOverflow3

[PASS] testMulWadUpOverflow3() (gas: 3788)
Logs:
  r 680564733841876926927
```

Recommended Mitigation

Remove the incorrect line.

```
function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256 z) {
    /// @solidity memory-safe-assembly
    assembly {
        // Equivalent to `require(y == 0 || x <= type(uint256).max / y)`.
        if mul(y, gt(x, div(not(0), y))) {
            mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
            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))
    }
}
```

Previous test passes, confirming that the mitigation is correct.

```
forge test --mt testMulWadUpOverflow2
...

[PASS] testMulWadUpOverflow2() (gas: 681)
```

Medium

[M-1] Incorrect custom error signature in mulWad()

Relevant GitHub Links

https://github.com/Cyfrin/2024-01-math-master/blob/84c149baf09c1558d7ba3493c7c4e68d83e7b3aa/src/MathMaster
```

Summary

The mulWad() function has a condition where it reverts with a custom error, but the returned signature is incorrect.

Vulnerability Details

According to the documentation, the custom error should be MathMasters__MulWadFailed() with signature 0xa56044f7, but the returned signature is 0xbac65e5b, which corresponds to MulWadFailed()

Also, the memory location where it is being stored is incorrect (0x40), as the revert is returning the last 4 bytes of the first word, so the returned signature is finally 0x00000000.

```
function mulWad(uint256 x, uint256 y) internal pure returns (uint256 z)
    {
    // @solidity memory-safe-assembly
    assembly {
        // Equivalent to `require(y == 0 || x <= type(uint256).max / y)
            `.
        if mul(y, gt(x, div(not(0), y))) {
@>              mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed
    ()`.
            revert(0x1c, 0x04)
        }
        z := div(mul(x, y), WAD)
    }
}
```

Impact

Dapps or contracts that interact with the contract implementing this library will get incorrect information about the error, and may malfunction.

Tools Used

Foundry, Manual review

Proof of Concept

This is a test that triggers the revert and expects the custom error signature 0xa56044f7.

```
function testMulWadRevert() public {
    uint256 x = type(uint256).max / 2;
    uint256 y = 100;
    vm.expectRevert(0x00000000);
    uint256 r = MathMasters.mulWad(x, y);
}
```

Test passes, confirming that the returned signature is empty.

```
forge test --mt testMulWadRevert
```

```
...

[PASS] testMulWadRevert() (gas: 3197)
```

Recommended Mitigation

Correct the custom error and memory location.

```
function mulWad(uint256 x, uint256 y) internal pure returns (uint256 z)
    {
    // @solidity memory-safe-assembly
    assembly {
        // Equivalent to `require(y == 0 || x <= type(uint256).max / y)
            `.
        if mul(y, gt(x, div(not(0), y))) {
-           mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
+           mstore(0x00, 0xa56044f7) // `MathMasters__MulWadFailed()`.
            revert(0x1c, 0x04)
        }
        z := div(mul(x, y), WAD)
    }
}
```

[M-2] Incorrect custom error signature in mulWadUp()

Relevant GitHub Links

https://github.com/Cyfrin/2024-01-math-master/blob/84c149baf09c1558d7ba3493c7c4e68d83e7b3aa/src/MathMaster

Summary

The mulWadUp() function has a condition where it reverts with a custom error, but the returned signature is incorrect.

Vulnerability Details

According to the documentation, the custom error should be MathMasters__MulWadFailed() with signature 0xa56044f7, but the returned signature is 0xbac65e5b, which corresponds to MulWadFailed()

Also, the memory location where it is being stored is incorrect (0x40), as the revert is returning the last 4 bytes of the first word, so the returned signature is finally 0x00000000.

```
function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256
    z) {
    /// @solidity memory-safe-assembly
    assembly {
        // Equivalent to `require(y == 0 || x <= type(uint256).max / y)
            `.
        if mul(y, gt(x, or(div(not(0), y), x))) {
```

```
@>              mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed
    ()`.
             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

Dapps or contracts that interact with the contract implementing this library will get incorrect information about the error, and may malfunction.

Tools Used

Foundry, Manual review

Proof of Concept

This is a test that triggers the revert and expects the custom error signature 0xa56044f7.

```
function testMulWadUpRevert() public {
    uint256 x = type(uint256).max / 2;
    uint256 y = 100;
    vm.expectRevert(0x00000000);
    uint256 r = MathMasters.mulWadUp(x, y);
    assertEq(r, (x * y - 1) / 1e18 + 1);
}
```

Test passes, confirming that the returned signature is empty.

```
forge test --mt testMulWadRevert
...

[PASS] testMulWadUpRevert() (gas: 3197)
```

Recommended Mitigation

Correct the custom error and memory location.

```
function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256
    z) {
    /// @solidity memory-safe-assembly
    assembly {
        // Equivalent to `require(y == 0 || x <= type(uint256).max / y)
            `.
        if mul(y, gt(x, div(not(0), y))) {
-           mstore(0x40, 0xbac65e5b) // `MathMasters__MulWadFailed()`.
+           mstore(0x00, 0xa56044f7) // `MathMasters__MulWadFailed()`.
```

```
8                revert(0x1c, 0x04)
9            }
10           if iszero(sub(div(add(z, x), y), 1)) { x := add(x, 1) }
11           z := add(iszero(iszero(mod(mul(x, y), WAD))), div(mul(x, y),
                 WAD))
12       }
13   }
```

Low

[L-1] Contracts using Solidity version 0.8.3 will not compile

Relevant GitHub Links

https://github.com/Cyfrin/2024-01-math-master/blob/84c149baf09c1558d7ba3493c7c4e68d83e7b3aa/src/MathMaster

Summary

The version pragma indicates that the library will be valid for versions 0.8.3 or higher and below 0.9.0, but it is not compatible with 0.8.3.

Vulnerability Details

The library is using custom errors, but that functionality was introduced in 0.8.4, so it will not compile with 0.8.3.

```
1  // SPDX-License-Identifier: MIT
2  // @notice We intentionally want to leave this as floating point so
       others can use it as a library.
3  pragma solidity ^0.8.3;
```

Impact

Contracts that require Soldity 0.8.3 will not be able to use this library.

Tools Used

Manual review

Proof of Concept

Set the compiler version to 0.8.3 in the tests and try to run.

MathMasters.t.sol

```
1  // SPDX-License-Identifier: MIT
2  // @notice We intentionally want to leave this as floating point so
       others can use it as a library.
3  pragma solidity 0.8.3;
```

```
forge test
...

Compiler run failed:
Error (2314): Expected ';' but got '('
  --> src/MathMasters.sol:14:41:
   |
14 |     error MathMasters__FactorialOverflow();
   |                                         ^
```

Recommended Mitigation

Change the version pragma to ^0.8.4 and indicate this change in the documentation.

MathMasters.sol

```
// SPDX-License-Identifier: MIT
// @notice We intentionally want to leave this as floating point so
   others can use it as a library.
-pragma solidity ^0.8.3;
+pragma solidity ^0.8.4;
```