Dria Swan Protocol Audit Report

Version 1.0

0xGondar

November 23, 2024

Dria Swan Audit Report

0xGondar

Oct 25th, 2024 => Nov 1st, 2024

Prepared by: 0xGondar Lead Auditors:

- 0xGondar

Table of Contents

Protocol Summary

Swan is structured like a market, where buyers are AI agents. By setting parameters like backstory, behavior, and objective you define how your agent will act. By using the budget you deposit, the agent buys the best items listed based on how aligned they are with its parameters. With each new asset bought, the agent's state changes and the simulation evolves.

Disclaimer

The 0xGondar 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

Audit contest triage by Codehawks. Contest page can be found here => https://codehawks.cyfrin.io/c/2024-10-swan-dria

Scope

```
contracts/
|-- libraries/
|   `-- Statistics.sol
|-- llm/
|   |-- LLMOracleCoordinator.sol
|   |-- LLMOracleManager.sol
|   |-- LLMOracleRegistry.sol
|   `-- LLMOracleTask.sol
`-- swan/
```

Issues found

-Severity	Number of issues found
High	1
Total	1

Findings

High

https://codehawks.cyfrin.io/c/2024-10-swan-dria/s/339

[H-1]Statistics Library Integer Overflow/Underflow in Variance Calculation renders LLMOracleCoordinator::validate unusable

Summary

There's an Integer Overflow/Underflow problem in Statistics::variance function which renders the critical standard deviation calculation (function stddev()) unusable. Due to this issue, LLMOracleCoordinator::validate function cannot be trusted.

Vulnerability Details

variance function in Statistics.sol contract cannot be used because it encounters with overflow/underflow error even with reasonable numbers.

```
    /// @notice Compute the variance of the data.
    /// @param data The data to compute the variance for.
    function variance(uint256[] memory data) internal pure returns (uint256 ans, uint256 mean) {
        mean = avg(data);
        uint256 sum = 0;
        for (uint256 i = 0; i < data.length; i++) {
            uint256 diff = data[i] - mean;
            sum += diff * diff;
        }
        ans = sum / data.length;
    }

    /// @notice Compute the standard deviation of the data.
    /// @dev Computes variance, and takes the square root.
    /// @param data The data to compute the standard deviation for.
    function stddev(uint256[] memory data) internal pure returns (uint256 ans, uint256 mean) {
        (uint256 _variance, uint256 _mean) = variance(data);
        mean = _mean;
        ans = sqrt(_variance);
    }
```

Note that variance is a key component of stddev function. To see the vulnerability in action, integrate Foundry to the project, create foundry/Statistics.t.sol inside the already existing test folder, and copy the following contents inside:

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity ^0.8.20;

import "forge-std/Test.sol";
import "forge-std/console.sol";
import "../../contracts/libraries/Statistics.sol";

contract StatisticsTest is Test {
    // Test 1: Known values test
    //!does not pass
    function test_stddev_known_values() public {
        uint256[] memory data = new uint256[]();
        data[0] = 2;
        data[1] = 4;
        data[2] = 4;
        data[3] = 4;
        data[4] = 5;
        data[5] = 5;
```

```
        data[6] = 7;
        data[7] = 9;

        (uint256 sd, uint256 mean) = Statistics.stddev(data);
        assertEq(mean, 5); // Mean should be 5
    }
    //! fails with panic: arithmetic underflow or overflow

    function test_stddev_known_values_easy() public {
        uint256[] memory data = new uint256[]();
        data[0] = 2;
        data[1] = 4;
        data[2] = 6;

        (uint256 sd, uint256 mean) = Statistics.stddev(data);
        assertEq(mean, 4); // Mean should be 4
        assertEq(sd, 2); // Standard deviation should be sqrt(4) = 2
    }

    //! fails with  panic: arithmetic underflow or overflow
    function test_variance_known_values() public {
        uint256[] memory data = new uint256[]();
        data[0] = 2;
        data[1] = 4;
        data[2] = 6;
        (uint256 ans, uint256 mean) = Statistics.variance(data);
        assertEq(mean, 4);
        console.log("ans", ans);
        console.log("mean", mean);
    }
}
```

Then run `forge test -vvvv --match-contract StatisticsTest` this will be the terminal output:

```
Ran 3 tests for test/foundry/Statistics.t.sol:StatisticsTest
[FAIL: panic: arithmetic underflow or overflow (0x11)]
    test_stddev_known_values() (gas: 2733)
Traces:
  [2733] StatisticsTest::test_stddev_known_values()
   => [Revert] panic: arithmetic underflow or overflow (0x11)

[FAIL: panic: arithmetic underflow or overflow (0x11)]
    test_stddev_known_values_easy() (gas: 1472)
Traces:
  [1472] StatisticsTest::test_stddev_known_values_easy()
   => [Revert] panic: arithmetic underflow or overflow (0x11)

[FAIL: panic: arithmetic underflow or overflow (0x11)]
    test_variance_known_values() (gas: 1398)
Traces:
```

```
  [1398] StatisticsTest::test_variance_known_values()
   => [Revert] panic: arithmetic underflow or overflow (0x11)


Failing tests:
Encountered 3 failing tests in test/foundry/Statistics.t.sol:
   StatisticsTest
[FAIL: panic: arithmetic underflow or overflow (0x11)]
   test_stddev_known_values() (gas: 2733)
[FAIL: panic: arithmetic underflow or overflow (0x11)]
   test_stddev_known_values_easy() (gas: 1472)
[FAIL: panic: arithmetic underflow or overflow (0x11)]
   test_variance_known_values() (gas: 1398)

Encountered a total of 3 failing tests, 0 tests succeeded
```

Impact

The impact is high because Statistics::stddev is used in LLMOracleCoordinator::finalizeValidation function that is called by LLMOracleCoordinator::validate. This is a critical part of the protocol, and cannot afford to fail in any way.

Tools Used

Foundry

Recommendations

Reconsider the Statistics::variance function or consider using fixed-point arithmetic library like PRB-Math