QUANTAMM
Security Review

DECEMBER, 2024

www.chaindefenders.xyz
https://x.com/ChDefendersEth

Lead Auditors

PeterSR

0x539.eth

Table of Contents

Protocol Summary

QuantAMM is a next generation DeFi protocol launching Blockchain Traded Funds (BTFs). LPs are no longer only chasing swap fees: the weights of the pool change to take advantage of current underlying price movements and therefore can overcome MEV and Impermanent Loss. QuantAMM does this in a continuous, responsive way with advanced, fully on-chain TradFi-style strategies.

Disclaimer

The ChainDefenders 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

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

Audit Details

Scope

Id	Files in scope
1	ChainlinkOracle.sol
2	MultiHopOracle.sol
3	QuantAMMStorage.sol
4	QuantAMMWeightedPoolFactory.sol
5	QuantAMMWeightedPool.sol
6	AntimomentumUpdateRule.sol
7	QuantammBasedRuleHelpers.sol
8	QuantammCovarianceBasedRule.sol
9	QuantammGradientBasedRule.sol
10	QuantammMathGuard.sol
11	QuantammMathMovingAverage.sol
12	QuantammVarianceBasedRule.sol
13	ChannelFollowingUpdateRule.sol
14	DifferenceMomentumUpdateRule.sol
15	MinimumVarianceUpdateRule.sol
16	MomentumUpdateRule.sol
17	PowerChannelUpdateRule.sol
18	UpdateRule.sol
19	UpdateWeightRunner.sol
20	LPNFT.sol
21	UpliftOnlyExample.sol
22	IQuantAMMWeightedPool.sol
23	IUpdateRule.sol
24	IUpdateWeightRunner.sol
25	OracleWrapper.sol

Roles

Id	Roles
1	Admin
2	User

Executive Summary

Issues found

Severity	Count	Description
High	1	Critical vulnerabilities
Medium	2	Significant risks
Low	0	Minor issues with low impact
Informational	0	Best practices or suggestions
Gas	0	Optimization opportunities

Findings

High

[HIGH-01] Missing not divisible by two

Summary

In `_calculateQuantAMMVariance` the `notDivisibleByTwo` if case is not added in the else case. In this situation logic will be not correct and it will calculate wrong results.

Vulnerability Details

During the calculations in `_calculateQuantAMMVariance`, there are two situations where lambda array is with 1 length and lambda array is with more than 1 elements. If the length is 1, everything is calculated correctly, even for array with odd number of elements due to this logic.

```
if (locals.notDivisibleByTwo) {
  unchecked {
      --locals.nMinusOne; // ok
  }
}

 ...

if (locals.notDivisibleByTwo) {
  unchecked {
      ++locals.nMinusOne;
  }
  locals.intermediateState =
      locals.convertedLambda.mul(locals.intermediateVarianceState[locals.nMinusOne]) +
      (_newData[locals.nMinusOne] - _poolParameters.movingAverage[locals.n + locals.nMinusOne])
          .mul(_newData[locals.nMinusOne] - _poolParameters.movingAverage[locals.nMinusOne])
          .div(TENPOWEIGHTEEN); // p(t) - p̄(t - 1))_i * (p(t) - p̄(t))_i

  locals.intermediateVarianceState[locals.nMinusOne] = locals.intermediateState;
  locals.finalState[locals.nMinusOne] = locals.oneMinusLambda.mul(locals.intermediateState);
  intermediateVarianceStates[_poolParameters.pool][locals.storageIndex] = locals
      .intermediateVarianceState[locals.nMinusOne];
}
```

The same logic should be used in the else case when the len of lambdas is more than 1. It is partially the same, but the first if statement is missing. This will lead to incorrect computations.

Impact

Logic will revert in the second if stament, due to element out of bound and not correct logic.

Tools Used

Manual review

Recommendations

Add the following code into the begging of the else case:

```
if (locals.notDivisibleByTwo) {
  unchecked {
      --locals.nMinusOne;
  }
}
```

Medium

[MEDIUM-01] DoS Due To Modification Of `UpdateWeightRunner`

Summary

The current implementation of `UpdateWeightRunner` introduces a critical vulnerability in the protocol. If the `quantammAdmin` modifies the `UpdateWeightRunner`, it could lead to unexpected behavior where the protocol breaks. Specifically:

1. A new `UpdateWeightRunner` might have a different `quantammAdmin`, which would not align with the existing `Pool`.
2. The rule required by the new `UpdateWeightRunner` is not set because the rule is defined during the `Pool` initialization phase.

This issue creates inconsistencies in the protocol, potentially leading to a denial of service (DoS) for affected pools.

Vulnerability Details

The vulnerability arises when the UpdateWeightRunner is changed, causing critical issues:

1. Admin Ownership Mismatch: The new UpdateWeightRunner may have a different quantAdmin, leading to conflicting authority and governance inconsistencies.
2. Missing Rules: The pool's rules, set during initialization, are not carried over to the new UpdateWeightRunner. This prevents updates, effectively causing a denial-of-service (DoS) for the pool.

Proof of Concept (POC)

Add the following test to `QuantAMMWeightedPool2TokenTest` to simulate the issue:

Initialization of the New `UpdateWeightRunner`:

```
updateWeightRunner1 = new MockUpdateWeightRunner(owner, addr2, false); // Add this to the constructor
```

POC Test Case:

```
MockUpdateWeightRunner updateWeightRunner1;

function
    testQuantAMMWeightedPoolGetNormalizedWeightsInitial_andThenChangeUpdateWeightRunn
    () public {
     QuantAMMWeightedPoolFactory.NewPoolParams memory params = _createPoolParams();
     params._initialWeights[0] = 0.6e18;
     params._initialWeights[1] = 0.4e18;

     (address quantAMMWeightedPool, ) = quantAMMWeightedPoolFactory.create(params);

     uint256[] memory weights = QuantAMMWeightedPool(quantAMMWeightedPool).getNormalizedWeights();

     int256;
     newWeights[0] = 0.6e18;
     newWeights[1] = 0.4e18;
     newWeights[2] = 0e18;
     newWeights[3] = 0e18;

     uint64;
     lambdas[0] = 0.2e18;

     int256;
     parameters0] = 0.2e18;

     address[][] memory oracles oracles[0][0]acle);

     MockMomentumRule momentumRule = new MockMomentumRule(owner);

     // Change UpdateWeightRunner
     vm.prank(owner);
     QuantAMMWeightedPool(quantAMMWeightedPool).
    setUpdateWeightRunnerAddress(address(updateWeightRunner1));

     QuantAMMWeightedPool(quantAMMWeightedPool).initialize(
         newWeights,
         IQuantAMMWeightedPool.PoolSettings(
             new IERC20 ,
             IUpdateRule(momentumRule),        oracles,
             60,
             lambdas,
             0.2e18,
             0.2e18,
             0.2e18,
             parameters,
             address(0)
         ),
         newWeights,
         newWeights,
         10
     );

     // Perform an update with the new runner
     vm.prank(owner);
     updateWeightRunner1.setApprovedActionsForPool(quantAMMWeightedPool
    , 1);
     updateWeightRunner1.performUpdate(quantAMMWeightedPool);
 }
```

Impact

Changing the `UpdateWeightRunner` leads to the following issues:

1. Denial of Service (DoS):
   The new `UpdateWeightRunner` does not inherit the rule for the existing pool, rendering it non-functional.
2. Unauthorized Updates:
   The `quantAdmin` of the initial `UpdateWeightRunner` can update the pool with the new `UpdateWeightRunner`, creating further inconsistencies.

These flaws disrupt the protocol and can lead to operational outages or malicious misuse.

Tools Used

Manual Review

Recommendations

To address this vulnerability, update the `setUpdateWeightRunnerAddress` function to synchronize `quantammAdmin` and ensure the rule is correctly set during the update. Modify the function as follows:

Updated Code

```
function setUpdateWeightRunnerAddress(address _updateWeightRunner)
    external override {
     require(msg.sender == quantammAdmin, "ONLYADMIN");
     updateWeightRunner = UpdateWeightRunner(_updateWeightRunner);
+    quantammAdmin = updateWeightRunner.quantammAdmin();
+    _setRule(); // Call set rule with the correct parameters
     emit UpdateWeightRunnerAddressUpdated(address(updateWeightRunner),
      _updateWeightRunner);
}
```

[MEDIUM-02] Wrong Uplift Fee Take

Summary

In `UpdateWeightRunner` we should have both `quantAMMUpliftFeeTake` and `quantAMMSwapFeeTake`. However, currently that is not the case.

Vulnerability Details

Setting `quantAMMUpliftFeeTake` is currently setting `quantAMMSwapFeeTake` and fetching it is fetching also the swap fee take which should not be the case.

Impact

Confusion of the fee structure and also wrong accounting of `quantAMMSwapFeeTake` when it is changed from `setQuantAMMUpliftFeeTake`.

Tools Used

Manual Review

Recommendations

Add the `quantAMMUpliftFeeTake` and fix the functionality of setting it and fetching it.