MetaSoccer Token Bonding Curve

Contracts Security Audit

Audit Resources:

Github Repository of the project was provided. (Link)

Project Author:

- Alex Fiestas (Github)
- Diego (Github)

Project Auditor:

- Umair Mirza (Github)
- Vinh Le (Github)

Audit Manager:

- Pradhumna Pancholi (GitHub)

Table of Contents

Audit Summary

The MetaSoccer Token Bonding Curve project has been compiled, deployed and tested using the **Hardhat** smart contract development tool chain and deployed on Polygon. The project is comprised of four smart contracts, namely:

- BatchedBancorMarketMaker.sol
- ERC20Token.sol
- TestDAI.sol
- BancorFormula.sol

Following libraries and interfaces have been integrated with the smart contracts:

- OpenZeppelin
- Bancor Market Interface

The contracts have been audited by 2 residents from October 24th to October 28th.

Scope

The scope of this audit is limited to the smart contracts mentioned above. Frontend modules of the project have not been audited.
The commit that has been audited is: **d5f1797d76e8d6196516995d234e40c054c87ffa**
This audit is about identifying potential vulnerabilities in the smart contracts. The audit may not identify all potential attack vectors or areas of vulnerability.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Access control is appropriate for this contract
Compiler	Medium	Solidity 0.8.9 is used which is quite recent and should not be trusted to deploy on mainnet at the moment
Complexity	Medium	● Similar variable names(shadowing) create a bit of confusion while reviewing. This way of writing code can be prone to mistakes later on. ● BatchedBancorMarketMaker.sol exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet.
Libraries	Good	Openzeppelin has been used for this project which is considered mature and stable libraries.
Decentralization	Good	The contract follows the rules of decentralization
Code Stability	Good	The last commit to the repository was on 20th March 2022 which suggests that the repository code is stable.
Documentation	Good	The project documentation is upto the standards.
Monitoring	Good	Events have been added to all important functions in the contract

Findings Description

Findings have been broken down into sections by their respective impact:

- Critical, High, Medium, Low Impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas Savings
    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

TestDAI.sol

Informational Findings

1. Informational - Solidity version 0.8.9 is not recommended for deployment

Proof of Concept

- Inside hardhat.config.js, the solidity version being used for deployment is 0.8.9.

```
module.exports = {
  solidity: {
    version: "0.8.9",
    settings: {
      optimizer: {
        enabled: true,
        runs: 100,
      },
    },
  },
```

Impact

Using a relatively untrusted version of Solidity can produce undesired side effects.

Recommendation

Consider deploying with:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.16

ERC20Token.sol

Low Findings

2. Low - Some variables are being shadowed

Proof of Concept

ERC20Token.constructor(address,uint256,string,string).name (contracts/ERC20Token.sol#31) shadows:
- ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64) (function)
- IERC20Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#17) (function)

ERC20Token.constructor(address,uint256,string,string).symbol (contracts/ERC20Token.sol#31) shadows:
- ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72) (function)
- IERC20Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#22) (function)

Impact

Variable shadowing can cause unnecessary confusion for the reviewer / tester. Sometimes the confusion can cause mistaken use of the variable which can end up in an error.

Recommendation

Rename the local variables that shadow another component.

Gas Optimization Findings

3. Optimization - Public function that could be declared external

Proof of Concept

mint(address,uint256) should be declared external:
- ERC20Token.mint(address,uint256) (contracts/ERC20Token.sol#47-51)

pause() should be declared external:

- ERC20Token.pause() (contracts/ERC20Token.sol#62-65)
unpause() should be declared external:
- ERC20Token.unpause() (contracts/ERC20Token.sol#76-79)

Impact

public functions that are never called by the contract should be declared external, and its immutable parameters should be located in calldata to save gas.

Recommendation

Use the external attribute for functions never called from the contract, and change the location of immutable parameters to calldata to save gas.

BancorFormula.sol

Medium Findings

4. Medium - Divide before multiply

Proof of Concept

Mutiple calculations are done with division made before multiplication. For example:

> BancorFormula.optimalLog(uint256) (contracts/bancor-formula/BancorFormula.sol#456-484) performs a multiplication on the result of a division:
>
> -x = x * FIXED_1 / 0xd3094c70f034de4b96ff7d5b6f99fcd8 (contracts/bancor-formula/BancorFormula.sol#463)
> -x = x * FIXED_1 / 0xa45af1e1f40c333b3de1db4dd55f29a7 (contracts/bancor-formula/BancorFormula.sol#464)

Impact

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation

Consider ordering multiplication before division.

5. Medium - Uninitialized local variables

Proof of Concept

BancorFormula.optimalExp(uint256).y (contracts/bancor-formula/BancorFormula.sol#500) is a local variable never initialized
BancorFormula.optimalLog(uint256).y (contracts/bancor-formula/BancorFormula.sol#459) is a local variable never initialized

Impact

Uninitialized local variables

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Gas Optimization Findings

6. Optimization - State variables that could be declared constant

Proof of Concept

BancorFormula.version (contracts/bancor-formula/BancorFormula.sol#12) should be constant

Impact

Constant state variables should be declared constant to save gas.

Recommendation

Add the constant attributes to state variables that never change.

7. Optimization - Public function that could be declared external

Proof of Concept

calculatePurchaseReturn(uint256,uint256,uint32,uint256) should be declared external:
- BancorFormula.calculatePurchaseReturn(uint256,uint256,uint32,uint256) (contracts/bancor-formula/BancorFormula.sol#187-205)
- IBancorFormula.calculatePurchaseReturn(uint256,uint256,uint32,uint256) (contracts/bancor-formula/interfaces/IBancorFormula.sol#8)

calculateSaleReturn(uint256,uint256,uint32,uint256) should be declared external:

- BancorFormula.calculateSaleReturn(uint256,uint256,uint32,uint256)
(contracts/bancor-formula/BancorFormula.sol#221-244)
- IBancorFormula.calculateSaleReturn(uint256,uint256,uint32,uint256)
(contracts/bancor-formula/interfaces/IBancorFormula.sol#9)
calculateCrossConnectorReturn(uint256,uint32,uint256,uint32,uint256) should be declared external:
- BancorFormula.calculateCrossConnectorReturn(uint256,uint32,uint256,uint32,uint256)
(contracts/bancor-formula/BancorFormula.sol#539-541)

Impact

public functions that are never called by the contract should be declared external, and its immutable parameters should be located in calldata to save gas.

Recommendation

Use the external attribute for functions never called from the contract, and change the location of immutable parameters to calldata to save gas.

Informational Findings

8. Informational - Some variables are being shadowed

Proof of Concept

BancorFormula.generalExp(uint256,uint8)
(contracts/bancor-formula/BancorFormula.sol#405-443) uses literals with too many digits:
- res += xi * 0x3442c4e6074a82f1797f72ac0000000
(contracts/bancor-formula/BancorFormula.sol#409)
BancorFormula.generalExp(uint256,uint8)
(contracts/bancor-formula/BancorFormula.sol#405-443) uses literals with too many digits:
- res += xi * 0x116b96f757c380fb287fd0e40000000
(contracts/bancor-formula/BancorFormula.sol#410)

Impact

Variable shadowing can cause unnecessary confusion for the reviewer / tester. Sometimes the confusion can cause mistaken use of the variable which can end up in an error.

Recommendation

Rename the local variables that shadow another component.

BatchedBancorMarketMaker.sol

Medium Findings

9. Medium - Division before multiplication

Proof of Concept

Mutiple functions use calculations where division is made before multiplication. For example (Line 424 - Line 426:

```
    function _currentBatchId() internal view returns (uint256) {
        return (block.number.div(batchBlocks)).mul(batchBlocks);
    }
```

Impact

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation

Consider ordering multiplication before division.

10. Medium - Dangerous strict equality is being used

Proof of Concept

BatchedBancorMarketMaker._slippageIsValid(BatchedBancorMarketMaker.Batch) (contracts/BatchedBancorMarketMaker.sol#483-493) uses a dangerous strict equality:
- staticPricePPM == 0 (contracts/BatchedBancorMarketMaker.sol#488)

```
    function _slippageIsValid(Batch storage _batch) internal view returns
(bool) {
        uint256 staticPricePPM = _staticPricePPM(_batch.supply,
_batch.balance, _batch.reserveRatio);
        uint256 maximumSlippage = _batch.slippage;

        // if static price is zero let's consider that every slippage is
valid
        if (staticPricePPM == 0) {
            return true;
        }
```

Impact

Complex calculations may not return accurate results sometimes. If the strict equality fails, the contract will be susceptible to failures.

Recommendation

Use an approach that does not use strict equality comparison.

Informational Findings

11. Informational - Variable names too similar

Proof of Concept

Variable BatchedBancorMarketMaker.removeCollateralToken(address)._collateral (contracts/BatchedBancorMarketMaker.sol#242) is too similar to BatchedBancorMarketMaker.collaterals (contracts/BatchedBancorMarketMaker.sol#94)

Impact

Similar variables make it really hard to review the code as it can create some confusion while reading.

Recommendation

Prevent variables from having similar names.