code4rena

Moxie

Smart Contract Security Assessment

Version 2.0

Audit dates: Jun 24 — Jun 28, 2024

Audited by: Alex the Entreprenerd
Hickuphh3

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About Moxie

Moxie is an orchestration of several smart contracts that can be executed via Frames, Actions and Apps/Clients. It represents foundational technology that anyone can use to add economic incentives to their Farcaster experience.

2.2 Scope

Repository	moxie-protocol/contracts
Commit Hash	850c4d98837568b364c45d090c6e4a392650166b
Mitigation Hash	41e30311c29dba520c0e420b86037fbe7925b88a

2.3 Audit Timeline

DATE	EVENT
Jun 24, 2024	Audit start
Jun 28, 2024	Audit end
Nov 28, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Informational	10
Total Issues	13

3. Findings Summary

ID	DESCRIPTION	STATUS
L-1	A collection of Footguns and Admin Risks	Resolved

L-2	`Vault.deposit` allows to donate to reserves without paying a fee	Resolved
L-3	Swaps lack deadline, which can result in paying an unintended fee	Resolved
I-1	Deviation between expected and real bought / sold amounts	Resolved
I-2	`AUCTION_ROLE` is inconsistently named	Resolved
I-3	Redundancies	Resolved
I-4	Spelling errors	Resolved
I-5	Certain checks can be stricter	Resolved
I-6	`subjectToken` burn process can be simplified	Resolved
I-7	QA: Unnecessary `msg.sender == address(this)` check in `EasyAuction`	Resolved
I-8	Informational: Quantitative Analysis of BancorFormula and Vault Donations	Resolved
I-9	Informational: Differential Invariant Testing of BancorFormula	Resolved
I-10	Purchases are safe from overflow provided they are within 2^96	Resolved

4. Findings

4.1 Low Risk

A total of 3 low risk findings were identified.

[L-1] A collection of Footguns and Admin Risks

Severity: Low	Status: Resolved

Context:

- moxie-protocol/contracts

Description:

Updating formula could be sandwiched, and can open up the project to arbitrage

The formula can be changed via a setter, in the case in which said change is queued, MEV actors may identify an opportunity to purchase cheap tokens and re-sell them at a profit via the new formula.

It's also important to keep in mind that changing the formula may change the relation between bought SubjectTokens and donation to reserves

Recommendation:

We recommend thoroughly testing changes to the BancorFormula for unintended consequences

Initial Deposit could be rounded down to zero via donation

After initializing the SubjectBondingCurve, the SubjectFactory will purchase tokens on behalf of `_subject` as a means to distribute fees to it

The purchase will be done with a 0 out check, due to this, along with the fact that Vault allow for donations means that a donation may be performed as means to cause this purchase to result in 0 shares out

You can check and alter the fuzz test: `test_checkLimits` to explore various ways in which the shares out can round down to zero

Recommendation:

We recommend ensuring that prices are relatively sane, for example enforcing a min purchase of 1e6 tokens will avoid most rebases

Will round down to zero but is protected by min price

```
  function test_roundDownProtocolBuy(uint256 buyAmount, uint256
sellAmount, uint256 amt) public {
      assertTrue(buyAmount * amt / sellAmount > 0);
  } // b * a << s -> Rounds down to zero
```

Min price as well as min amount need to prevent rebase risk See: `test_checkLimits`

```
function test_checkLimits(uint64 collateral) public {
    // vm.assume(newTs > 1e18 && newTs < 1e24);
    // vm.assume(newDeposits > 1e18 && newDeposits < 1e24);
    // vm.assume(collateral > 1e6);
    totalSupply = 1;
    deposits = 1e18;
    assertTrue(viewBuy(collateral) == 0, "Not zero");
}
```

```
Failing tests:
Encountered 1 failing test in
test/recon/CryticToFoundry.sol:CryticToFoundry
[FAIL. Reason: Not zero; counterexample:
calldata=0x27351fe50000000000000000000000000000000000000000f69b0f0e8300e69
91a17b81b10000000000000000000000000000000fffffffffffffffffffffffffffff
f0000000000000000000000000000000000000000000000000000000000517e7dc args=
[122113227234052180590717170526 5 [1.221e30],
340282366920938463463374607431768211455 [3.402e38], 85452764 [8.545e7]]]
test_checkLimits(uint128,uint128,uint128) (runs: 0, μ: 0, ~: 0)

[FAIL. Reason: Not zero; counterexample:
calldata=0x27351fe50000000000000000000000000000000000000000000000055cd04
692939881b0000000000000000000000000000000000000004ba4bd451ca8bc5df07
20000000000000000000000000000000000000000000000000000000000008c4b args=
[6182602713159272475 [6.182e18], 357216390581869359263858 [3.572e23],
35915 [3.591e4]]] test_checkLimits(uint128,uint128,uint128) (runs: 17, μ:
128462, ~: 158904)
```

Will round down to zero provided a small total supply and a high amount of deposits

```
    function test_checkLimits(uint256 collateral) public {
        vm.assume(collateral < uint256(10_000_000_000) * 1e18);
```

```
        vm.assume(collateral < 1e22);
        // vm.assume(newTs > 1e18 && newTs < 1e24);
        // vm.assume(newDeposits > 1e18 && newDeposits < 1e24);
        // vm.assume(collateral > 1e6);
        totalSupply = 1;
        deposits = 1e22;
        assertTrue(viewBuy(collateral) == 0, "Not zero");
    }
```

Auction may be spammed with small orders:

- Mitigated by min size

Make all logic initializable

Prevents any risk with UUPS being self-destructable

Suggested Upgrade Pattern

- ProxyAdmin -> Owned by Timelock -> Owned by Multisig
- Proxies

Suggested Post-deployment Checks

- Contracts are deployed, verified and initialized (both logic and proxy)
- Script to check Proxy admin is timelock

Easy Auction Must have 0 fees

- There must be no fees set in `EasyAuction`; `SubjectFactory` is incompatible with non-zero fees https://github.com/moxie-protocol/ido-contracts/blob/add_subject_factory_check/contracts/EasyAuction.sol#L175-L181

URI can never be changed

- `MoxiePass` URI setter isn't exposed; the minter must set it correctly upon minting (essentially URI is immutable for each NFT)

Role changing must be behind a timelock

- More crucial roles to pay attention to when granting: `Vault.TRANSFER_ROLE`, `TokenManager.MINT_ROLE`, `SubjectFactory.ONBOARDING_ROLE`, `MoxieBondingCurve.UPDATE_FORMULA_ROLE`

Recommendation:

Moxie: Fixed with PR-26.

Zenith: Verified

[L-2] `Vault.deposit` allows to donate to reserves without paying a fee

Severity: Low	Status: Resolved

Context:

- Vault.sol#L50-L68

Description: `MoxieBondingCurve` setup an intended path to donate to reserves, which consists of:

- Buying shares
- With onBehalf set to `address(0)`

Vault.deposit has no access controls, it allows anyone to deposit more `token` to any subjectToken, this will cause the value of the token to rebase, without paying fees

During the review, we spent some time looking into ways to abuse the donation mechanism, we believe that a rebase can be used to steal some value, but that as long as rational values are set (See https://github.com/code-423n4/2024-06-moxie-pro-league/issues/23), then no particular risk is introduced beside the loss of fees to the protocol

(See https://github.com/code-423n4/2024-06-moxie-pro-league/issues/13)

Recommendation: Either document this additional token flow, or add access control to the Vault as a means to ensure that donations can only be performed via `buyShares`

Moxie: Fixed with PR-26.

Zenith: Verified.

[L-3] Swaps lack deadline, which can result in paying an unintended fee

Severity: Low	Status: Resolved

Context:

- MoxieBondingCurve.sol#L530-L535
- MoxieBondingCurve.sol#L569-L574

Description: `MoxieBondingCurve` allows buying and selling shares, it also allows the `UPDATE_FEES_ROLE` to `updateFees`

Due to this, an order that may be intended at a time, may be unintended at another time

Adding a deadline, would enforce that an order is either filled in time or discarded by the sequencer

Recommendation:

Consider adding a deadline to `buyShares` and `sellShares`

Moxie: Acknowledged

4.2 Informational

A total of 10 informational findings were identified.

[I-1] Deviation between expected and real bought / sold amounts

Severity: Informational	Status: Resolved

Context:

- MoxieBondingCurve.sol#L705-L774

Description: View methods `calculateTokensForBuy()` and `calculateTokensForSell()` were added to estimate the moxie token amounts required to purchase / received when selling subject tokens. Equivalency between the estimated amount and actual amount used / received for the specific amounts was checked.

A reasonable amount range for 18 decimal tokens is assumed: `[1e14, 100_000_000e18]`. Empirically via fuzzing, a maximum deviation of 0.1% was found.

It is important to note that this deviation becomes increasingly large for very small reserve ratios.

Recommendation: Small reserve ratios should be avoided, and users should be aware that there may be slight discrepancies between the estimated and actual amounts.

Moxie: Fixed with PR-43.

Zenith: Verified

[I-2] `AUCTION_ROLE` is inconsistently named

Severity: Informational	Status: Resolved

Context:

- SubjectFactory.sol#L21

Description: `AUCTION_ROLE` is a permissioned role that is given access to update the `auctionDuration` and `auctionOrderCancellationDuration`. Its naming is inconsistent with other roles that have the `UPDATE` prefix that update other parameters.

Recommendation: Rename `AUCTION_ROLE` to `UPDATE_AUCTION_ROLE`.

Moxie: Renamed AUCTION_ROLE -> UPDATE_AUCTION_ROLE. Fixed with PR 26.

Zenith: Verified

[I-3] Redundancies

Severity: Informational	Status: Resolved

Context:

- MoxieBondingCurve.sol#L38
- SubjectFactory.sol#L315-L317
- MoxieBondingCurve.sol#L521-L522

Description: The defined custom error is unused.

The zero value check on `newSupplyToMint` is redundant as it will be checked in `tokenManager.mint()`.

Finally, the TODOs are no longer applicable as they've been resolved.

Recommendation:

```
- error MoxieBondingCurve_InvalidReserveFactory();

- if (newSupplyToMint == 0) {
-   revert SubjectFactory_BuyAmountTooLess();
- }

- //todo add moxie pass check
- //todo decide if onBehalfOf can be address(0)
```

Moxie: Fixed with PR-26

Zenith: Verified

[I-4] Spelling errors

Severity: Informational	Status: Resolved

Context:

- MoxieBondingCurve.sol#L94
- MoxieBondingCurve.sol#L285
- MoxieBondingCurve.sol#L478
- SubjectFactory.sol#L279-L311
- SubjectFactory.sol#L349
- TokenManager.sol#L28
- Vault.sol#L20

Description: The referenced lines have spelling errors or TODOs that should be removed for clarity.

Recommendation:

```
- represeny
+ represent

- recieved
+ received

- reseve
+ reserve

- _clearningOrder
+ _clearingOrder

- transaaction
+ transaction

- Implemetation
+ Implementation

- Intialize
+ Initialize
```

Moxie: Fixed with PR-26

Zenith: Verified.

[I-5] Certain checks can be stricter

Severity: Informational	Status: Resolved

Context:

- MoxieBondingCurve.sol#L200-L205
- SubjectFactory.sol#L90-L93
- SubjectFactory.sol#L481-L484
- SubjectFactory.sol#L98-L102
- SubjectFactory.sol#L498-L502

Description: The fees are individually checked to not exceed `PCT_BASE`. The more effective check would be to ensure that the total fees charged do not exceed `PCT_BASE`.

For auction time updates, there should be a sanity check to ensure that `auctionOrderCancellationDuration` cannot exceed `auctionDuration`.

Recommendation:

```
if (
  !_feeIsValid(_feeInput.protocolBuyFeePct + _feeInput.subjectBuyFeePct)
||
  !_feeIsValid(_feeInput.protocolSellFeePct +
_feeInput.subjectSellFeePct)
) revert MoxieBondingCurve_InvalidFeePercentage();

////////////////////////////////////////////

if (
  !_feeIsValid(_feeInput.protocolFeePct + _feeInput.subjectFeePct)
) revert SubjectFactory_InvalidFeePercentage();

////////////////////////////////////////////

if (_auctionOrderCancellationDuration == 0)
  revert SubjectFactory_InvalidAuctionOrderCancellationDuration();

// this ensures that _auctionDuration >=
_auctionOrderCancellationDuration > 0
if (_auctionDuration < _auctionOrderCancellationDuration)
  revert SubjectFactory_InvalidAuctionDuration();
```

Moxie: Fixed with PR-26

Zenith: Verified

[I-6] `subjectToken` burn process can be simplified

Severity: Informational	Status: Resolved

Context:

- MoxieBondingCurve.sol#L390-L391

Description: `MoxieBondingCurve` transfers the `subjectToken` from the user to itself, then burns it. There is a simpler function `burnFrom()` that allows direct burning of the user's token, with allowance given.

Recommendation:

```
- _subjectToken.safeTransferFrom(msg.sender, address(this), _sellAmount);
- _subjectToken.burn(_sellAmount);
+ _subjectToken.burnFrom(msg.sender, _sellAmount);
```

Moxie: Fixed with PR-26

Zenith: Verified

[I-7] QA: Unnecessary `msg.sender == address(this)` check in `EasyAuction`

Severity: Informational	Status: Resolved

Context: EasyAuction.sol#L470-L473

Description:

The function will use a jump when called by `settleAuctionAtomically` so it shouldn't be necessary to have it

```
        require(
            subjectFactory == msg.sender  || msg.sender == address(this),
            "Caller is not the subject factory address"
        );
```

Recommendation:

Delete `msg.sender == address(this)`

Moxie: Acknowledged

[I-8] Informational: Quantitative Analysis of BancorFormula and Vault Donations

Severity: Informational	Status: Resolved

Context:

- BancorFormula.sol

Description:

The following is a quantitative analysis done on the behaviour of the formula, specifcally looking at how donations may be used to attack it

While rebasing risk is present, no major risk was found from allowing donations of reserves

Sheet:
https://docs.google.com/spreadsheets/u/6/d/1CFRKI0ZOpzg0v0nXgr_YJOBCzZ4EX5yKFgJU2G6JdkQ/edit?usp=sharing

Script used to generate the sheet:

```
// SPDX-License-Identifier: GPL-2.0
pragma solidity ^0.8.0;

import {Test} from "forge-std/Test.sol";
import {TargetFunctions} from "./TargetFunctions.sol";
import {FoundryAsserts} from "@chimera/FoundryAsserts.sol";
import "forge-std/console2.sol";

contract CryticToFoundry is Test, TargetFunctions, FoundryAsserts {
    function setUp() public {
        setup();

    }

    uint256 totalSupply = 1e18; // Subject
    uint256 deposits = 1e18; // Token
    uint256 donations = 0; // Token

    uint256 ONE = 1e18;
    uint32 reserveRatio = 660000; // .66

    function test_checkLimits(uint256 collateral) public {
        vm.assume(collateral < uint256(10_000_000_000) * 1e18);

        vm.assume(collateral < 1e22);
```

```
        // vm.assume(newTs > 1e18 && newTs < 1e24);
        // vm.assume(newDeposits > 1e18 && newDeposits < 1e24);
        // vm.assume(collateral > 1e6);
        totalSupply = 1;
        deposits = 1e22;
        assertTrue(viewBuy(collateral) == 0, "Not zero");
    }

    function test_roundDownProtocolBuy(uint256 buyAmount, uint256
sellAmount, uint256 amt) public {
        assertTrue(buyAmount * amt / sellAmount > 0);
    } // b * a << s -> Rounds down to zero

    function viewBuy(uint256 amt) internal returns (uint256 shares) {
        return target.calculatePurchaseReturn(totalSupply, deposits,
reserveRatio, amt);
    }
    function viewSell(uint256 shares) internal returns (uint256 amt) {
        return target.calculateSaleReturn(totalSupply, deposits,
reserveRatio, shares);
    }

    function buy(uint256 amt) internal returns (uint256 shares) {
        uint256 newShares = viewBuy(amt);
        deposits += amt;
        totalSupply += newShares;

        return newShares;
    }
    function sell(uint256 shares) internal returns (uint256 amt) {
        uint256 amt = viewSell(shares);
        deposits -= amt;
        totalSupply -= shares;

        return amt;
    }

    function donate(uint256 amt) internal {
        donations += amt;
        deposits += amt;
    }

    function donateMultple(uint256 multiple) internal {
        donations += deposits * multiple;
        deposits += deposits * multiple;
    }
```

```
function test_buyPrices() public {
    _logHeaders();
    uint256 snapshot = vm.snapshot();
    // Start at 1e18;
    _logState(ONE);

    uint256 increment = 1e18; // 1 tokens per time
    uint256 count = 100;
    for(uint256 i; i < count; i++) {
        // Compare donation vs deposit
        _compareDepositVSDonation(increment);

        // Do a deposit (move the story)
        buy(increment);
    }
    _logState(ONE);
}

function test_sellPrices() public {
    _logHeaders();

    uint256 count = 10;
    uint256 amt = 1e18 / count;

    _logState(amt);

    for(uint256 i; i < count - 1; i++) {
        sell(amt);
        _logState(amt);
    }
}

function test_buyAndThenSellPrice() public {
    _logHeaders();

    uint256 amt = 1e18 / 2;

    _logStateConsole(amt);
    uint256 amtToSell = viewBuy(amt);
    buy(amt);
    _logStateConsole(amt);
    sell(amtToSell);
    _logStateConsole(amt);
}
```

```
    function _compareDepositVSDonation(uint256 amt) internal {
        uint256 snapshot = vm.snapshot();

        // Compare deposit vs Donation to price
        buy(amt);
        _logState(ONE);
        vm.revertTo(snapshot);

        donate(amt);
        _logState(ONE);
        vm.revertTo(snapshot);

        donateMultple(2);
        _logState(ONE);
        vm.revertTo(snapshot);
    }

    function _logHeaders() internal {
        console2.log("totalSupply", ",", "deposits", ",");
        console2.log("donations", ",", "viewBuy(ONE)");
        console2.log(",", "viewSell(ONE)");
    }
    function _logState(uint256 amt) internal {
        // Token Reserves
        // Total Supply
        // Price for next Buy
        // Price for next Sell
        console2.log(";");
        // console2.log("totalSupply", totalSupply);
        // console2.log("deposits", deposits);
        // console2.log("donations", donations);
        // console2.log("Buying with 1e18 tokens yields", viewBuy(ONE));
        // console2.log("Selling 1e18 shares yields", viewSell(ONE));
        console2.log(totalSupply, ",", deposits, ",");
        console2.log(donations, ",", viewBuy(amt));
        console2.log(",", viewSell(amt));
    }

    function _logStateConsole(uint256 amt) internal {
        // Token Reserves
        // Total Supply
        // Price for next Buy
        // Price for next Sell
        console2.log(";");
        console2.log("totalSupply", totalSupply);
        console2.log("deposits", deposits);
```

```
            console2.log("donations", donations);
            console2.log("Buying with amt tokens yields", viewBuy(amt));
            console2.log("Selling amt shares yields", viewSell(amt));
        }
    }
```

Recommendation: None

Moxie: Acknowledged

[I-9] Informational: Differential Invariant Testing of BancorFormula

Severity: Informational	Status: Resolved

Context:

- BancorFormula.sol

Description:

`BancorFormula` was fuzzed against the last known safe implementation (deployed by theGraph, compiled with Solidity 0.7.6)

The result is that after 100 Million Tests, all functions act in the same way as the reference

Run details: https://getrecon.xyz/shares/d5d96f05-3317-4f7d-822e-54967b8bc387

The full suite is available here (invite only): https://github.com/GalloDaSballo/moxie-bancor-differential

Recommendation: None

Moxie: Acknowledged

[I-10] Purchases are safe from overflow provided they are within 2^96

Severity: Informational	Status: Resolved

Context: SubjectFactory.sol#L282-L289

Description:

One of the main risks in initializing the bonding curve would be having to pay a price that is too burdensome

This can be avoided by ensuring that the Auction has rational minimum prices (See https://github.com/code-423n4/2024-06-moxie-pro-league/issues/23)

An additional risk would be overflow, which would cause a permanent denial of service. However, because EasyAuction only works with up to 2^96 tokens and the `moxie` token has a total supply that is below that, the following test shows that the math is sound

Safe from overflow

```
    function testOverflows(uint96 buyAmount, uint96 sellAmount, uint256
amount) public {
        amount = bound(amount, 0,  type(uint128).max);
        sellAmount = uint96(bound(sellAmount, 1, type(uint96).max));
        uint256 res = buyAmount * amount / sellAmount;
    }
```

Recommendation: Considering that the Moxie token will have a total supply of 1e28 -> log_2(1e28) == 93.0139866568, no change is necessary

Moxie: Acknowledged