code4rena

# Moxie

## Smart Contract Security Assessment

Version 1.0

| | |
|---|---|
| Audit dates: | Dec 01 — Dec 04, 2024 |
| Audited by: | peakbolt<br>10xhash |

## 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/tree/feature/referral-rewards |
| Commit Hash | 459b2e0377e88980ce58b2d9a91bc212d4bf8a69 |

## 2.3 Audit Timeline

| DATE | EVENT |
|---|---|
| Dec 01, 2024 | Audit start |
| Dec 04, 2024 | Audit end |
| Dec 04, 2024 | Report published |

## 2.4 Issues Found

| SEVERITY | COUNT |
|---|---|
| Critical Risk | 0 |
| High Risk | 1 |
| Medium Risk | 0 |
| Low Risk | 0 |
| Informational | 3 |
| Total Issues | 4 |

## 3. Findings Summary

| ID | DESCRIPTION | STATUS |
|---|---|---|
| H-1 | \`updateReserveRatio()\` allows users to profit from risk-free trade | Resolved |
| I-1 | Malleable signatures should be rejected in \`withdrawWithSig()\` | Resolved |

| | | |
|---|---|---|
| I-2 | Typo in SubjectFactoryV2.sol | Resolved |
| I-3 | Unnecessary used of `whenNotPaused` modifier | Resolved |

## 4. Findings

### 4.1 High Risk

A total of 1 high risk findings were identified.

#### [H-1] `updateReserveRatio()` allows users to profit from risk-free trade

| Severity: High | Status: Resolved |
|---|---|

**Context:**

- MoxieBondingCurveV2.sol#L625-L647

**Description:** `updateReserveRatio()` allows DAO to update reserve ratio for a specific subject. This will change the bonding curve parameters and affect the purchase/sale price of the tokens.

That means users can sandwich `updateReserveRatio()` to profit from a risk-free trade.

An attack scenario could occur as follows,

1. DAO decided to call `updateReserveRatio()` to lower the reserve ratio from 500k ppm to 400k ppm for subject `X`. This will make the subject token price higher after the change.
2. Attacker anticipates change in reserve ratio by DAO and proceeds to sandwich the `updateReserveRatio()` with a `buyShares()` before the call and a `sellShares()` after the call.
3. The attacker realized a profit buying the subject tokens at a lower price and selling them at a higher price due to the change in reserve ratio.

```
    function updateReserveRatio(
        address _subject,
        uint32 _newReserveRatio
    ) external onlyRole(UPDATE_RESERVE_RATIO) {

        uint32 currentReserveRatio = reserveRatio[_subject];

        if (currentReserveRatio == 0) {
            revert MoxieBondingCurve_SubjectNotInitialized();
        }

        if ( _newReserveRatio == 0 ||
!_reserveRatioIsValid(_newReserveRatio)) {
```

```
            revert MoxieBondingCurve_InvalidReserveRation();
        }

        reserveRatio[_subject] = _newReserveRatio;

        emit SubjectReserveRatioUpdated(
            _subject,
            currentReserveRatio,
            _newReserveRatio
        );
    }
```

**Recommendation:** This can be fixed by introducing a function that pauses the buy/sell shares for the specific subject token before calling `updateReserveRatio()` after one or more blocks has past.

This prevents anyone from anticipating the change by buying/selling the tokens before the reserve ratio is updated. Also, a specific pause for the subject will not affect trading of other subject tokens.

```
+    mapping(address subject => uint32 _paused) public lastPaused;

+  // modifier to be used for buy/sell shares
+   modifier whenNotTradingPaused() {
+        if (paused[_subject] != 0) revert
MoxieBondingCurve_TradingPaused();
+        _;
+   }


+   function pauseTrading(address _subject, bool _pause) external
onlyRole(UPDATE_RESERVE_RATIO) {
+        if(_pause)
+            lastPaused[_subject] = block.timestamp;
+        else
+            lastPaused[_subject] = 0;
+   }

    function updateReserveRatio(
        address _subject,
        uint32 _newReserveRatio
    ) external onlyRole(UPDATE_RESERVE_RATIO) {
+       //only allow reserve ratio change when subject is paused for at
least a window period of 5 secs
```

```
+       if (paused[_subject] == 0 || paused[_subject] + 5 >=
block.timestamp)
              revert MoxieBondingCurve_SubjectNotPaused();

        uint32 currentReserveRatio = reserveRatio[_subject];

        if (currentReserveRatio == 0) {
            revert MoxieBondingCurve_SubjectNotInitialized();
        }

        if ( _newReserveRatio == 0 ||
!_reserveRatioIsValid(_newReserveRatio)) {
            revert MoxieBondingCurve_InvalidReserveRation();
        }

        reserveRatio[_subject] = _newReserveRatio;

        emit SubjectReserveRatioUpdated(
            _subject,
            currentReserveRatio,
            _newReserveRatio
        );
    }


+    // whenNotTradingPaused modifier for buy and selling of shares
   function buySharesFor(
     ...
-   whenNotPaused
+  whenNotPaused whenNotTradingPaused
    ...

   function buyShares(
     ...
-   whenNotPaused
+  whenNotPaused whenNotTradingPaused
    ...

   function sellShares(
     ...
-   whenNotPaused
+  whenNotPaused whenNotTradingPaused
    ...

   function sellShares(
     ...
```

```diff
-    whenNotPaused
+   whenNotPaused whenNotTradingPaused
     ...

    function buySharesForV2(
      ...
-    whenNotPaused
+   whenNotPaused whenNotTradingPaused
     ...

    function buySharesV2(
      ...
-    whenNotPaused
+   whenNotPaused whenNotTradingPaused
     ...

    function sellSharesForV2(
      ...
-    whenNotPaused
+   whenNotPaused whenNotTradingPaused
     ...

    function sellSharesV2(
      ...
-    whenNotPaused
+   whenNotPaused whenNotTradingPaused
     ...
```

**Moxie:** Fixed in the following commits: @27eb8dee9236.. & @392134cb5dd0...

Keeping this simple. Only adding check that trading should be paused. For delay we will rely on off chain DAO process.

**Zenith:** Verified. Resolved with recommended pauseTrading() with delay check done off-chain.

## 4.2 Informational

A total of 3 informational findings were identified.

### [I-1] Malleable signatures should be rejected in `withdrawWithSig()`

| | |
|---|---|
| **Severity:** Informational | **Status:** Resolved |

**Context:**

- ProtocolRewards.sol#L210

**Description:** `withdrawWithSig()` uses `ecrecover()` directly to verify the signer for withdrawal of rewards.

It is not recommended to do so as `erecover()` allows for malleable (non-unique) signatures. Due to symmetric structure of elliptic curve, there exists two valid signatures for the same hash because for every set of v, r, s there is a mirror set of v, r, s.

Using `ecrecover()` directly makes it possible for an attacker to compute another valid signature for the same hash without the private key.

```
    function withdrawWithSig(
        address from,
        address to,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external whenNotPaused IfNonBlocked(from) {
        if (block.timestamp > deadline) {
            revert PROTOCOL_REWARDS_SIGNATURE_DEADLINE_EXPIRED();
        }

        ...
        bytes32 digest = _hashTypedDataV4(withdrawHash);

>>>     address recoveredAddress = ecrecover(digest, v, r, s);
```

**Recommendation:** This can be resolved by using OpenZeppelin's `ECDSA.erecover()`, which ensures unique signatures by only accepting signature with an s-value in the lower half order.

Note that the use of nonce has mitigated the risk of signature replay attack due to this issue, so the fix is recommended as a best practice.

**Moxie:** Fixed with the following commit - @27eb8dee9236..

**Zenith:** Verified.

## [I-2] Typo in SubjectFactoryV2.sol

| Severity: Informational | Status: Resolved |
|---|---|

**Context:**

- SubjectFactoryV2.sol#L448

**Description:** `token` is incorrect and should be removed

```
     * @param _subject Address of subject token.
```

**Recommendation:** Change to `@param _subject Address of subject`

**Client:** Fixed in the following commit

**Zenith:** Verified

## [I-3] Unnecessary used of `whenNotPaused` modifier

| Severity: Informational | Status: Resolved |
|---|---|

**Context:**

- MoxieBondingCurveV2.sol#L469-L475

**Description:** The `whenNotPaused` modifier is unnecessary for the `_sellSharesInternal` function as it is already applied to all the invoking external functions

```
    function _sellSharesInternal(
        address _subject,
        uint256 _sellAmount,
        address _onBehalfOf,
        uint256 _minReturnAmountAfterFee,
        address _orderReferrer
    ) internal whenNotPaused returns (uint256 returnAmount_) {
```

**Recommendation:** Remove the modifier

**Client:** Fixed in the following commit

**Zenith:** Verified