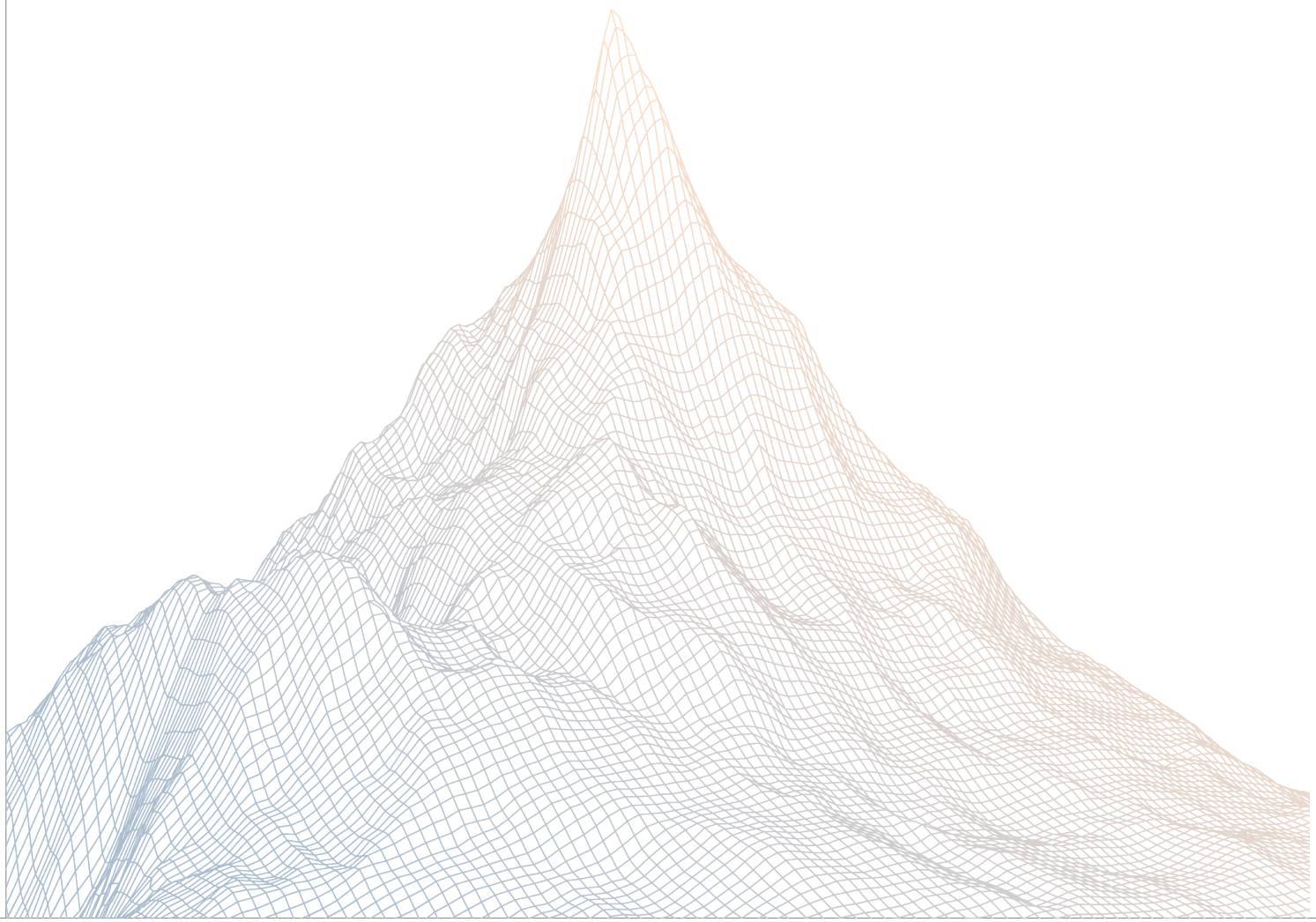
Zenith

Moxie Protocol

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: February 2nd to February 5th, 2025

AUDITED BY: carrotsmuggler
ladboy

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 Protocol

Moxie is a community-owned and community-governed Farcaster protocol. Our mission is to grow the Farcaster GDP.

The core protocol team, from Airstack, have been building on Farcaster for nearly two years. We believe that Farcaster provides a fantastic tableau for next generation social networks.

2.2 Scope

The engagement involved a review of the following targets:

Target	2025-02-moxie
Repository	https://github.com/zenith-security/2025-02-moxie
Commit Hash	`1a07acd4945259388ebdc57a1936e6e7953235c5`

Files	`MoxieBondingCurveV2.sol`
	`GraduationHook.sol`
	`FakeDonator.sol`

2.3 Audit Timeline

DATE	EVENT
February 18, 2025	Audit start
February 21, 2025	Audit end
February 26, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	2
Informational	1
Total Issues	**5**

3

Findings Summary

ID	DESCRIPTION	STATUS
H-1	User can suffer from slippage loss when excessive fund that exceeds graduation cap is used to swap for subject token	Acknowledged
M-1	Uniswap v4 Trading pool cannot be created if the MOXIE deposit amount from function ‘itializeSubjectBondingCurve ‘ exceed the graduation cap	Resolved
L-1	‘calculateTokensForBuy‘ doesnt take into account graduation	Resolved
L-2	Moxie left in contract can be stolen	Resolved
I-1	Error message should be swapped to indicate fee setting when distributing the fee	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] User can suffer from slippage loss when excessive fund that exceeds graduation cap is used to swap for subject token

SEVERITY: High	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: High

Target:

- MoxieBondingCurveV2

Description:

When user's deposit triggers a graduation, the excessive fund (remainder fund) is swapped from MOXIE token to the subject token in newly created pool.

```
if (remainder > 0) {
    uint256 remainingMinAmountOut = _minReturnAmountAfterFee < shares_ ?
    0 :  _minReturnAmountAfterFee - shares_;
    // we have a remainder, therefore graduate the subject and swap the
    remainder on the liquidity pool
    _graduateSubject(_subject, remainder, _onBehalfOf,
    remainingMinAmountOut);
}
```

However, the `_minReturnAmountAfterFee` parameter serves as a slippage protection to ensure the user pays the MOXIE token to receive at least the amount of subject token when buying shares using Bancor formula.

The issue is that the `_minReturnAmountAfterFee` value was estimated based on the Bancor formula, based on the bonding curve. Multiple users will be participating in the bonding so users cannot exactly predict whether their transaction will make the token graduate or not. If the token graduates, then the slippage criteria estimated from the Bancor formula will be used for a uniswap swap, which can lead to unintended amounts of slippage.

If `remainingMinAmountOut` is 0 and there is still a remainder amount, a swap is executed with no slippage protection.

The uniswap model has the price increase faster then the Bancor model, since the uniswap model follows a hyperbolic curve. So in a lot of cases, the slippage protection will be too tight for a uniswap swap and will lead to reverted transactions.

Essentially, using the slippage from the Bancor formula for uniswap swaps can lead to multiple issues.

Recommendations:

Consider refunding the additional MOXIE token to the user instead of using the additional MOXIE token to execute a swap. Continuing the swap with the same formula can lead to higher than expected slippage.

Moxie: We don't think this is an issue because the correct minimum amount considering both, the bonding curve formula and the swap can be correctly calculated off chain

Zenith: Acknowledged.

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Uniswap v4 Trading pool cannot be created if the MOXIE deposit amount from function `itializeSubjectBondingCurve` exceed the graduation cap

`SEVERITY:` Medium	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` Low

Description:

The subject factory needs to finalize the auction and call `initializeSubjectBondingCurve` to initialize the bonding curve.

The moxie token is transferred from factory and deposit to the vault.

Consider the action below:

1. initializeSubjectBondingCurve is called, the MOXIE reserved amount exceed the graduation cap.

2. The token can be graduated, but the function `graduateSubject` is not called inside `initializeSubjectBondingCurve`

3. The subjectFee is not 0, after the subject curve is initialized, the code tries to buy shares (paying MOXIE for subject token)

```
moxieBondingCurve.initializeSubjectBondingCurve(
    _subject, _reserveRatio, _bondingSupply, bondingAmount_,
    _platformReferrer
);

uint256 subjectFeeInSubjectToken = 0;
///@dev Instead of returning subject fee to subject, give buy subject shares
    for subject.
if (subjectFee_ > 0) {
    token.approve(address(moxieBondingCurve), subjectFee_);
    subjectFeeInSubjectToken = moxieBondingCurve.buySharesFor(
        _subject,
        subjectFee_,
        _subject,
        0 //slippage settings not needed as this is first buy transaction.
    );
}
```

4. Transaction revert with error `MoxieBondingCurve_SubjectReadyForGraduation()` because when buying shares, the function `_whenNotGraduated` checks if the graduation is met, if the graduation cap is met, then buyShares cannot be called and transaction reverts.

```
_whenNotGraduated(_subject);
```

```
function _whenNotGraduated(address _subject) internal view {
    // check that the subject is not already graduated
    if (subjectGraduated(_subject))
    revert MoxieBondingCurve_SubjectAlreadyGraduated();
    // check that the subject is currently not above the graduation market
    cap after the contract upgrade
    uint32 subjectReserveRatio = reserveRatio[_subject];
    address subjectToken = tokenManager.tokens(_subject);
    uint256 currentReserves = vault.balanceOf(subjectToken, address(token));
    if (currentReserves
    >= _reservesRequiredForGraduation(subjectReserveRatio)) {
        revert MoxieBondingCurve_SubjectReadyForGraduation();
```

```
    }
}
```

Recommendations:

Calling `graduateSubject` in the function `initializeSubjectBondingCurve`, and not call `moxieBondingCurve.buySharesFor` in the factory contract.

Moxie: Resolved with @9e97f1de2...

Zenith: Verified.

4.3 Low Risk

A total of 2 low risk findings were identified.

[L-1] `calculateTokensForBuy` doesnt take into account graduation

`SEVERITY:` Low	`IMPACT:` Low
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target:

- MoxieBondingCurveV2

Description:

The `calculateTokensForBuy` function simulates the results of a token purchase from the bonding curve. However, the issue is that this function does not consider graduation. In case enough tokens are purchased, the token graduates and is added into a uniswap pool. Once that happens, the Bancor formula is not appropriate to calculate the swap results.

Thus if the token is close to graduation, this function can return erroneous values.

Recommendations:

The function is only accurate until the `_reservesRequiredForGraduation` limit is hit. After that It is inaccurate. Consider only returning values up to the Bancor formula limit, or also reporting this inaccuracy.

Moxie: Resolved with @194500bee7...

Zenith: Verified.

[L-2] Moxie left in contract can be stolen

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target:

- MoxieBondingCurveV2

Description:

Liquidity is added to the Uniswap V4 pool with specified amount0Max and amount1Max. This means that liquidity can be added for less than the amount of tokens present in the vault, and there can be remaining moxie tokens in the contract after the liquidity addition. In this case, the remaining moxie tokens in the contract after graduation can be stolen.

This can be done by calling the `distributeSwapFee` function. Here, after fees are collected from the uniswap position, all moxie tokens are regarded as fees, a part of which is paid to the subjects as well.

```
recipients[0] = feeBeneficiary;
amounts[0] = totalMoxieBalance * swapFeeRatioProtocolPct / PCT_BASE;
reasons[0] = bytes4(keccak256("SWAP_FEE"));

recipients[1] = _subject;
amounts[1] = totalMoxieBalance - amounts[0];
reasons[1] = bytes4(keccak256("PROTOCOL_FEE"));
```

This means that a part of the remaining moxie in the contract can be claimed by any subject by calling this fee function.

Recommendations:

Consider tracking the amount of tokens incoming from uniswap during the `DECREASE_LIQUIDITY` call, and only paying subjects a part of that. Any remaining moxie tokens can be swept into the `feeBeneficiary`. This will prevent random users from claiming the remaining moxie from the contract.

Moxie: Resolved with @05af8eaf1c...

Zenith: Verified.

4.4 Informational

A total of 1 informational findings were identified.

[I-1] Error message should be swapped to indicate fee setting when distributing the fee

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target:

- MoxieBondingCurveV2.sol

Description:

```
recipients[0] = feeBeneficiary;
amounts[0] = totalMoxieBalance * swapFeeRatioProtocolPct / PCT_BASE;
reasons[0] = bytes4(keccak256("SWAP_FEE"));

recipients[1] = _subject;
amounts[1] = totalMoxieBalance - amounts[0];
reasons[1] = bytes4(keccak256("PROTOCOL_FEE"));
```

When distributing the fee, the first fee is computed based on swapFeeRatioProtocolPct and it is the protocol fee. The second fee is the swap fee, the error message should be swapped to indicate fee setting.

Recommendations:

```
  recipients[0] = feeBeneficiary;
  amounts[0] = totalMoxieBalance * swapFeeRatioProtocolPct / PCT_BASE;
reasons[0] = bytes4(keccak256("SWAP_FEE"));
reasons[0] = bytes4(keccak256("PROTOCOL_FEE"));

  recipients[1] = _subject;
  amounts[1] = totalMoxieBalance - amounts[0];
reasons[1] = bytes4(keccak256("PROTOCOL_FEE"));
reasons[1] = bytes4(keccak256("SWAP_FEE"));
```

Moxie: Resolved by swapping the error message here.

Zenith: Verified.