ElasticSwap contest
Findings & Analysis Report

2022-03-02

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of ElasticSwap contest smart contract system written in Solidity. The code contest took place between January 20—January 26 2022.

Wardens

39 Wardens contributed reports to the ElasticSwap contest:

1. WatchPug (jtp and ming)
2. gzeon
3. 0x0x0x
4. Meta0xNull
5. danb
6. camden
7. hyh
8. Dravee
9. pauliax
10. 0x1f8b
11. harleythedog
12. OriDabush

13. Jujic
14. cmichel
15. rfa
16. UncleGrandpa925
17. yeOlde
18. Ruhum
19. p4st13r4 (0x69e8 and 0xb4bb4)
20. byterocket (pseudorandom and pmerkleplant)
21. robee
22. BouSalman
23. defsec
24. csanuragjain
25. wuwe1
26. sirhashalot
27. Tomio
28. 0v3rf10w
29. sorrynotsorry
30. bobi
31. solgryn
32. cccz
33. SolidityScan (cyberboy and zombie)
34. ckksec
35. egjlmn1

This contest was judged by Alex the Entreprenerd.

Final report assembled by liveactionllama and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 9 unique vulnerabilities and 49 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity, 1 received a risk rating in the category of MEDIUM severity, and 6 received a risk rating in the category of LOW severity.

C4 analysis also identified 13 non-critical recommendations and 27 gas optimizations.

Scope

The code under review can be found within the [C4 ElasticSwap contest repository](), and is composed of 3 smart contracts written in the Solidity programming language and includes 1120 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on [OWASP standards]().

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website]().

High Risk Findings (2)

[H-01] In the case of Single Asset Entry, new liquidity providers will suffer fund loss due to wrong formula of ΔRo

Submitted by WatchPug

Current Implementation

When `baseToken` rebase up

Per the document: https://github.com/ElasticSwap/elasticswap/blob/a90bb67e2817d892b517da6c1ba6fae5303e9867/ElasticSwapMath.md#:~:text=When%20there%20is%20alpha Decay

and related code: https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L227-L283

`Gamma` is the ratio of shares received by the new liquidity provider when `addLiquidity()` (ΔRo) to the new totalSupply (total shares = Ro’ = Ro + ΔRo).

```
ΔRo = (Ro/(1 - γ)) * γ

        Ro * Gamma
    = --------------
         1 - Gamma
⟺
ΔRo * ( 1 - Gamma ) = Gamma * Ro
ΔRo - Gamma * ΔRo = Gamma * Ro
ΔRo = Gamma * Ro + Gamma * ΔRo
           ΔRo
Gamma = ---------
         Ro + ΔRo
```

In the current implementation:

```
γ = ΔY / Y' / 2 * ( ΔX / α^ )
```

ΔY is the `quoteToken` added by the new liquidity provider. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L277

Y’ is the new Y after `addLiquidity()`, `Y' = Y + ΔY`. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L272
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L278

ΔX is `ΔY * Omega`. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L259-L263
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L279

α^ is `Alpha - X`. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L234-L235
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L280

For instance:

Given:

- Original State: X = Alpha = 1, Y = Beta = 1, Omega = X/Y = 1
- When `baseToken` rebase up: Alpha becomes 10
- Current State: Alpha = 10, X = 1, Y = Beta = 1, Omega = 1

When: new liquidity provider `addLiquidity()` with 4 quoteToken:

```
                4           4 * Omega      16
    Gamma = ------------ * ------------ = ----
             (1+4) * 2       10 - 1        90
```

After `addLiquidity()`:

- baseToken belongs to the newLP: 10 * 16 / 90 = 160 / 90 = 1.7777777777777777
- quoteToken belongs to the newLP: (1+4) * 16 / 90 = 80 / 90 = 0.8888888888888888
- In the terms of `quoteToken`, the total value is: 160 / 90 / Omega + 80 / 90 = 240 / 90 = 2.6666666666666665

As a result, the new liquidity provider suffers a fund loss of `4 - 240 / 90 = 1.3333333333333333 in the terms of quoteToken`

The case above can be reproduced by changing the numbers in [this test unit](#).

When `baseToken` rebase down

Per the document: https://github.com/ElasticSwap/elasticswap/blob/a90bb67e2817d892b517da6c1ba6fae5303e9867/ElasticSwapMath.md#:~:text=When%20there%20is%20betaDecay

and related code: https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L297-L363

`Gamma` is the ratio of shares received by the new liquidity provider when `addLiquidity()` (ΔRo) to the new totalSupply (total shares = Ro' = Ro + ΔRo).

```
    ΔRo = (Ro/(1 - γ)) * γ

            Ro * Gamma
        = ---------------
             1 - Gamma
⟺
    ΔRo * ( 1 - Gamma ) = Gamma * Ro
    ΔRo - Gamma * ΔRo = Gamma * Ro
    ΔRo = Gamma * Ro + Gamma * ΔRo
               ΔRo
    Gamma = ---------
             Ro + ΔRo
```

In the current implementation:

```
γ = ΔX / X / 2 * ( ΔXByQuoteTokenAmount / β^ )
```

ΔX is the amount of `baseToken` added by the new liquidity provider. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L357

X is the balanceOf `baseToken`. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L358

ΔXByQuoteTokenAmount is ΔX / Omega, the value of ΔX in the terms of `quoteToken`. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L318-L322
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L329-L333
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L359

β^ is maxΔX / Omega, the value of maxΔX in the terms of `quoteToken`. `maxΔX = X - Alpha`. See:

- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L304-L305
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L318-L322
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L341-L342
- https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L360

For instance:

Given:

- Original State: X = Alpha = 10, Y = Beta = 10, Omega = X/Y = 1
- When `baseToken` rebase down, Alpha becomes 1
- Current State: Alpha = 1, X = 10, Y = Beta = 10, Omega = 1

When: new liquidity provider `addLiquidity()` with `4 baseToken`

```
               4            4 / Omega          8
Gamma = -------- * ---------------- = ----
           10 * 2     (10-1) / Omega       90
```

After `addLiquidity()`:

- baseToken belongs to the newLP: (1 + 4) * 8 / 90 = 40 / 90 = 0.44444444444444444
- quoteToken belongs to the newLP: 10 * 8 / 90 = 80 / 90 = 0.88888888888888888

- In the terms of quoteToken, the total value is: 40 / 90 + 80 / 90 * Omega = 120 / 90 = 1.3333333333333333 < 4

As a result, the new liquidity provider suffers a fund loss of `4 - 120 / 90 = 2.6666666666666665 in the terms of quoteToken`

The case above can be reproduced by changing the numbers in this test unit.

The correct formula for ΔRo

See issue page for details.

Recommendation

Update code and document using the correct formula for ΔRo.

Oxean (ElasticSwap) confirmed and commented:

> Finding is valid - solution seems to be partially correct and we are working on the fully correct version.

> It seems that the suggested formula doesn't cover a rebase down correctly and this is where our efforts are focused now.

Alex the Entreprenerd (judge) commented:

> The warden has identified an issue with the math that reliably will provide a less-than-expected value to single-sided liquidity providers. The warden showed a consistent way for this to occur and while the recommended fix may not be completely correct, I believe the finding to be valid.

> Because the warden found a set of cases that reliably make the protocol return less value than expected when compared to the goals of the protocol, I believe High Severity to be appropriate.

Oxean (ElasticSwap) resolved

[H-02] Transferring `quoteToken` to the exchange pool contract will cause future liquidity providers to lose funds

Submitted by WatchPug

In the current implementation, the amount of LP tokens to be minted when `addLiquidity()` is calculated based on the ratio between the amount of newly added `quoteToken` and the current wallet balance of `quoteToken` in the `Exchange` contract.

However, since anyone can transfer `quoteToken` to the contract, and make the balance of `quoteToken` to be larger than `_internalBalances.quoteTokenReserveQty`, existing liquidity providers can take advantage of this by donating `quoteToken` and make future liquidity providers receive fewer LP tokens than expected and lose funds.

https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L578-L582

```
    liquidityTokenQty = calculateLiquidityTokenQtyForDoubleAssetEntr
        _totalSupplyOfLiquidityTokens,
        quoteTokenQty,
        _quoteTokenReserveQty // IERC20(quoteToken).balanceOf(addres
    );
```

PoC

Given:

- The `Exchange` pool is new;
- Alice `addLiquidity()` with `1e18 baseToken` and `1e18 quoteToken`, recived `1e18` LP token;
- Alice transfer `99e18 quoteToken` to the `Exchange` pool contract;
- Bob `addLiquidity()` with `1e18 baseToken` and `1e18 quoteToken`;
- Bob `removeLiquidity()` with all the LP token in balance.

Expected Results: Bob recived `1e18 baseToken` and >= `1e18 quoteToken`.

Actual Results: Bob recived ~ `0.02e18 baseToken` and ~ `1e18 quoteToken`.

Alice can now `removeLiquidity()` and recive ~ `1.98e18 baseToken` and ~ `100e18 quoteToken`.

As a result, Bob suffers a fund loss of `0.98e18 baseToken`.

Recommendation

Change to:

```
    liquidityTokenQty = calculateLiquidityTokenQtyForDoubleAssetEntr
        _totalSupplyOfLiquidityTokens,
        quoteTokenQty,
        _internalBalances.quoteTokenReserveQty
    );
```

Oxean (ElasticSwap) confirmed and commented:

> This does appear to be correct after attempting a POC. Thank you WatchPug!

Alex the Entreprenerd (judge) commented:

> The warden identified a way to exploit the protocol math to devalue future liquidity provision at the advantage of early liquidity providers.

> The exploit is extractive in nature, however, because this is reliably performable and effectively breaks the protocol's goals and mechanics, I believe High Severity to be appropriate.

Oxean (ElasticSwap) resolved

Medium Risk Findings (1)

[M-01] The value of LP token can be manipulated by the first minister, which allows the attacker to dilute future liquidity providers' shares

Submitted by WatchPug, also found by camden, danb, and hyh

For the first minter of an Exchange pool, the ratio of `X/Y` and the `totalSupply` of the LP token can be manipulated.

A sophisticated attacker can mint and burn all of the LP tokens but `1 Wei`, and then artificially create a situation of rebasing up by transferring baseToken to the pool contract. Then `addLiquidity()` in `singleAssetEntry` mode.

Due to the special design of `singleAssetEntry` mode, the value of LP token can be inflated very quickly.

As a result, `1 Wei` of LP token can be worthing a significate amount of baseToken and quoteToken.

Combine this with the precision loss when calculating the amount of LP tokens to be minted to the new liquidity provider, the attacker can turn the pool into a trap which will take a certain amount of cut for all future liquidity providers by minting fewer LP tokens to them.

https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L493-L512

```
    } else {
        // this user will set the initial pricing curve
        require(
            _baseTokenQtyDesired > 0,
            "MathLib: INSUFFICIENT_BASE_QTY_DESIRED"
        );
        require(
            _quoteTokenQtyDesired > 0,
            "MathLib: INSUFFICIENT_QUOTE_QTY_DESIRED"
        );

        tokenQtys.baseTokenQty = _baseTokenQtyDesired;
        tokenQtys.quoteTokenQty = _quoteTokenQtyDesired;
        tokenQtys.liquidityTokenQty = sqrt(
            _baseTokenQtyDesired * _quoteTokenQtyDesired
        );

        _internalBalances.baseTokenReserveQty += tokenQtys.baseToker
        _internalBalances.quoteTokenReserveQty += tokenQtys.quoteTok
    }
```

https://github.com/code-423n4/2022-01-elasticswap/blob/d107a198c0d10fbe254d69ffe5be3e40894ff078/elasticswap/src/libraries/MathLib.sol#L204-L212

```
    function calculateLiquidityTokenQtyForDoubleAssetEntry(
        uint256 _totalSupplyOfLiquidityTokens,
        uint256 _quoteTokenQty,
        uint256 _quoteTokenReserveBalance
    ) public pure returns (uint256 liquidityTokenQty) {
        liquidityTokenQty =
            (_quoteTokenQty * _totalSupplyOfLiquidityTokens) /
            _quoteTokenReserveBalance;
    }
```

PoC

Given:

- The `Pool` is newly created;
- The market price of `baseToken` in terms of `quoteToken` is `1`.

The attacker can do the following steps in one tx:

1. `addLiquidity()` with `2 Wei of baseToken` and `100e18 quoteToken`, received `14142135623` LP tokens;
2. `removeLiquidity()` with `14142135622` LP tokens, the Pool state becomes:
3. totalSupply of LP tokens: 1 Wei
4. baseTokenReserveQty: 1 Wei
5. quoteTokenReserveQty: 7071067813 Wei

6. `baseToken.transfer()` 7071067812 Wei to the Pool contract;
7. `addLiquidity()` with no baseToken and `50e18 quoteToken`;
8. `swapBaseTokenForQuoteToken()` with `600000000000000 baseToken`, the Pool state becomes:
9. totalSupply of LP tokens: 1 Wei
10. quoteTokenReserveQty 591021750159032
11. baseTokenReserveQty 600007071067801
12. `baseToken.transfer()` 999399992928932200 Wei to the Pool contract;
13. `addLiquidity()` with no baseToken and `1e18 quoteToken`, the Pool state becomes:
14. totalSupply of LP tokens: 1 Wei
15. quoteTokenReserveQty: 1000000000000000013
16. quoteTokenReserveQty: 985024641638342212
17. baseTokenDecay: 0

From now on, `addLiquidity()` with less than `1e18` of `baseToken` and `quoteToken` will receive `0` LP token due to precision loss.

The amounts can be manipulated to higher numbers and cause most future liquidity providers to receive fewer LP tokens than expected, and the attacker will be able to profit from it as the attacker will take a larger share of the pool than expected.

Recommendation

Consider requiring a certain amount of minimal LP token amount (eg, 1e8) for the first minter and lock some of the first minter's LP tokens by minting ~1% of the initial amount to the factory address.

Oxean (ElasticSwap) confirmed, but disagreed with High severity and commented:

> Thanks for the report, I don't agree with the severity based on this

> `From now on, addLiquidity() with less than 1e18 of baseToken and quoteToken will receive 0 LP token due to precision loss.`

> which in your example represents a user trying to add dust to the contract after the attack.

> I think we will implement the minimum locked liquidity to avoid rounding errors, but this attack assumes users are adding dust to the contract and that they are totally unaware of the contract state which is incorrect. Users specific a min and a max token qty's when adding liquidity.

> Would recommend med. risk on this one if not low risk given the attack is on "dust" amounts of tokens.

Alex the Entreprenerd (judge) changed severity to Medium and commented:

> I agree with the finding and remember coming across it when reading the Yearn V0.4.2 Audit by Trails of Bits.

> Ultimately this is contingent on a donation, that will make each share more valuable, so it's effectively a way to use rounding against making dust donations.

> Technically this same idea can be extended to huge donations, however there are very dubious economic reasons as to why you'd do that (perhaps frontrunning a moderate deposit with the goal of using this method to earn that MEV).

> Ultimately this is something that can happen anytime you have X shares and Y totalSupply If the total supply reaches greater units than the shares, then integer division will inevitably eat out some of those shares.

> Have yet to see a long term solution to this rounding problem, however, a simple initial addition that mints 1e18 shares will require some economic commitment by the potential exploiters.

> Agree with medium severity.

0xean (ElasticSwap) resolved

Low Risk Findings (6)

- [L-01] Inclusive conditions *Submitted by pauliax, also found by 0x1f8b, danb, harleythedog, and Jujic*

- [L-02] removeLiquidity() _tokenRecipient Lack of Zero Address Check May Cause User Lose Fund Permanently *Submitted by MetaOxNull*
- [L-03] Revert when K >= 2^256 *Submitted by gzeon*
- [L-04] Fee-on-transfer check can be avoided *Submitted by harleythedog, also found by 0x1f8b, cmichel, danb, and pauliax*
- [L-05] Description of `_expirationTimestamp` is not exact *Submitted by 0x0x0x*
- [L-06] Incorrect implementation of `_quoteTokenQtyMin`, `_baseTokenQtyMin` *Submitted by WatchPug*

Non-Critical Findings (13)

- [N-01] Math base functions can be made internal *Submitted by hyh*
- [N-02] 10 ** 18 can be changed to 1e18 *Submitted by Jujic*
- [N-03] Gas: `MathLib.sol` is importing `Exchange.sol` *Submitted by Dravee*
- [N-04] Exchange.sol is not Pausable *Submitted by yeOlde, also found by hyh*
- [N-05] Comment missing function parameter *Submitted by sirhashalot*
- [N-06] Leftover tokens will be stuck in the contract with no ways to recover *Submitted by hyh*
- [N-07] `ExchangeFactory.sol`'s `transferOwnership` should be a two-step process *Submitted by Dravee, also found by cccz and defsec*
- [N-08] createNewExchange() Possible to Add Elastic Token as Quote Token Due to No Validation *Submitted by MetaOxNull*
- [N-09] Use of Similar variable names *Submitted by SolidityScan*
- [N-10] Users can grief name and symbol for a market, DAO unable to change *Submitted by camden, also found by 0x1f8b, ckksec, cmichel, danb, defsec, egjlmn1, hyh, and UncleGrandpa925*
- [N-11] Base token properties not verified *Submitted by sirhashalot, also found by danb and sirhashalot*
- [N-12] Gas: `ExchangeFactory.feeAddress()` should be declared external *Submitted by Dravee*
- [N-13] swapBaseTokenForQuoteToken and swapQuoteTokenForBaseToken do not check output quantities to be achievable *Submitted by hyh*

Gas Optimizations (27)

- [G-01] Custom Errors *Submitted by sorrynotsorry, also found by 0v3rf10w, byterocket, defsec, Dravee, Jujic, Meta0xNull, robee, sirhashalot, WatchPug, and yeOlde*
- [G-02] Redundant `return` for named returns *Submitted by WatchPug, also found by Dravee, robee, and yeOlde*
- [G-03] Unchecked maths *Submitted by pauliax, also found by defsec, Dravee, gzeon, Jujic, OriDabush, Tomio, WatchPug, and yeOlde*
- [G-04] Initialize to default state is redundant *Submitted by WatchPug, also found by yeOlde*
- [G-05] Redundant code *Submitted by wuwe1, also found by csanuragjain, gzeon, Ruhum, and WatchPug*
- [G-06] Repeated calls *Submitted by pauliax, also found by defsec, Dravee, OriDabush, Ruhum, and sirhashalot*
- [G-07] Simplify `MathLib#sqrt()` can save gas *Submitted by WatchPug, also found by p4st13r4*
- [G-08] Gas Optimization: float multiplication optimization *Submitted by gzeon*
- [G-09] Gas Optimization: `> 0` is less efficient than `!= 0` for uint in require condition *Submitted by gzeon, also found by 0x0x0x, bobi, BouSalman, byterocket, defsec, Dravee, Jujic, Meta0xNull, robee, Ruhum, solgryn, and WatchPug*
- [G-10] using modifier instead of function can save gas *Submitted by rfa*
- [G-11] quoteTokenQtyToReturn = internalBalances.quoteTokenReserveQty *Submitted by pauliax*
- [G-12] Gas Optimization: Use deterministic contract address *Submitted by gzeon*
- [G-13] Shift Right instead of Dividing by 2 *Submitted by byterocket, also found by 0x0x0x, BouSalman, and Dravee*
- [G-14] Remove unused code can save gas *Submitted by WatchPug*
- [G-15] Cache and read storage variables from the stack can save gas *Submitted by WatchPug*
- [G-16] Outdated versions of OpenZeppelin library *Submitted by WatchPug*

- [G-17] Gas: Conditional flow optimization in `Exchange.sol:removeLiquidity()` *Submitted by Dravee*
- [G-18] Making the MathLib internal *Submitted by UncleGrandpa925*
- [G-19] `internalBalance` state variable is read and written multiple times within a single transaction *Submitted by Ruhum, also found by Dravee*
- [G-20] saving gas by not returning the variables that was declared to be returned *Submitted by OriDabush*
- [G-21] inlining a function to save gas *Submitted by OriDabush*
- [G-22] `removeLiquidity.sol#baseTokenQtyToRemoveFromInternalAccounting` should not be cached *Submitted by 0x0x0x*
- [G-23] Gas in `MathLib.sol:calculateQuoteTokenQty()` : SLOADs minimization *Submitted by Dravee*
- [G-24] Gas in `MathLib.sol:calculateQtyToReturnAfterFees()` : Avoid expensive calculation by checking if `_tokenASwapQty == 0 ||` `_tokenBReserveQty == 0` *Submitted by Dravee*
- [G-25] Gas: Mark `ExchangeFactory.sol:setFeeAddress()` as payable *Submitted by Dravee*
- [G-26] Gas: Reorder require statements `MathLib.sol:calculateAddLiquidityQuantities()` to save gas on revert *Submitted by Dravee*
- [G-27] Gas: Reorder require statements `Exchange.sol:removeLiquidity()` to save gas on revert *Submitted by Dravee*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

Top