STATE MIND

Buttonswap core

05-07-2023 - 11-08-2023

Table of contents

1. Project Brief 3

2. Finding Severity breakdown 4

3. Summary of findings 5

4. Conclusion 5

5. Findings report 6

High

- Rebase affects swap amounts 6
- Ability to swap tokens for free 7
- Potential loss of reservoir from arbitrage opportunities 9

Medium

- Imprecise value returned by _closestBound() 10
- The paramSetter address can block single side mints and burns 10
- Fund locking vulnerability via isPausedSetter role 11
- Possibility of manipulation over movingAveragePrice0 12
- Back-run/front-run of unpause 13
- DOS of mintWithReservoir and burnFromReservoir 13

Informational

Gas optimizations: Loops Improvement 14

Code style improvements 14

Gas optimizations: Caching Storage Variables 15

Gas optimizations: sendOrRefundFee modifier 15

Pool lock funds if one of the tokens with zeroed balance 15

Unbounded configuration parameters 16

ERC20 code duplication 16

No events in setters 16

Little gas optimizations 16

Possible manipulation of reservoirs by directly transferring tokens 17

movingAveragePrice0Last gas optimization 17

Minting and burning from reservoir don't have a slippage protection 17

Risk Associated with isPausedSetter Role 18

6. Appendix A. Linter 19

7. Appendix B. Slither 21

8. Appendix C. Tests 34

1. Project Brief

Title	Description
Client	Buttonwood
Project name	Buttonswap core
Timeline	05-07-2023 - 11-08-2023
Initial commit	7b4a64319b8232237f7682ef9773ed2dcd94ceb1
Final commit	807b6d73cb1e97d5776aa1dbdecb96754d72f98e

Project Scope

The audit covered the following files:

- ButtonswapFactory.sol
- PairMath.sol
- UQ112x112.sol
- Math.sol
- ButtonswapERC20.sol
- ButtonswapPair.sol
- IButtonswapPairEvents.sol
- IButtonswapPairErrors.sol
- IButtonswapPair.sol
- IButtonswapERC20Errors.sol
- IButtonswapERC20Events.sol
- IButtonswapERC20.sol
- IButtonswapCallee.sol
- IButtonswapFactoryEvents.sol
- IButtonswapFactory.sol
- IButtonswapFactoryErrors.sol

2. Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	3 (0 fixed, 3 acknowledged)
Medium	6 (3 fixed, 3 acknowledged)
Informational	13 (8 fixed, 5 acknowledged)
Total	22 (11 fixed, 11 acknowledged)

4. Conclusion

During the audit of Buttonswap core codebase, 22 issues were found in total:

- 3 high severity issues (3 acknowledged)
- 6 medium severity issues (3 fixed, 3 acknowledged)
- 13 informational severity issues (8 fixed, 5 acknowledged)

The final reviewed commit is 807b6d73cb1e97d5776aa1dbdecb96754d72f98e

Deployment

File name	Contract deployed on mainnet
ButtonswapFactory.sol	0xb8de4ab6c65e274630f5279f74eb69b66327ce50

5. Findings report

HIGH-01 | Rebase affects swap amounts | Acknowledged

Description

It is claimed that the protocol is resistant to a rebase of tokens. Namely, the price of the token swap remains unchanged after the rebase of the token in the pair. However, this is not the case, consider two examples.

The first one:

1. **pool0 = 1000000**
2. **pool1 = 1000000**
3. **reservoir0 = 0**
4. **reservoir1 = 100000** - **10%** from pool1

If effectively swap **y = 10000** token1 for **x** token0 then **x** would be **9871**.

However, let's assume that the **token1** rebase occurred before the swap, the rebase is **-18%**

1. **pool0 = 900000**
2. **pool1 = 900000**
3. **reservoir0 = 100000**
4. **reservoir1 = 0**

Then **x** would be **9860** with the same **y**.

If effectively swap **y = 10000** token1 for **x** token0 then **x** would be **9860**.

So, the resulting **x** differs from the previous one.

The second one:

1. **pool0 = 1000000**
2. **pool1 = 1000000**
3. **reservoir0 = 0**
4. **reservoir1 = 100000** - **10%** from pool1

If effectively swap **y = 10000** token1 for **x** token0 then **x** would be **9871**.

However, let's assume that the **token0** rebase occurred before the swap, the rebase is **5%**.

1. **pool0 = 1050000**
2. **pool1 = 1050000**
3. **reservoir0 = 0**
4. **reservoir1 = 50000**

If effectively swap **y = 10000** token1 for **x** token0 then **x** would be **9876**.

So, the resulting **x** differs from the previous one.

Recommendation

We recommend reviewing the logic of the active balance or explicitly mention the possibility of such situations

Client's comments

Updated documentation in PR: https://github.com/buttonwood-protocol/buttonswap-core/pull/100

HIGH-02	Ability to swap tokens for free	Acknowledged

Description

Functions **mintFromReservoir** and **burnFromReservoir** in **ButtonswapPair.sol** contract are calculating amount resulting from virtual swap without any fee for the next dual mint/burn.
Consider an example:

```
// all amounts were calculated with 10 ** 18 multiplier

A = 10_000_000; // Token0 active liquidity in pair
B = 10_000_000; // Token1 active liquidity in pair

// So, the price is 1-1

reservoir1 = B / 10;

// Let:

uint256 x = 500_000;

// User has ~1_000_000 (2 * x) Token0 and he want to get at least 500_000 (x) Token1

// He can make direct swap Token0->Token1, so:

// Using swap math:

uint256 x = PairMathExtended.getSwapOutputAmount(AmountNeeded, vars.pool0, vars.pool1);

// by the formula:
// x * (A * 1000 + AmountNeeded * 997) = B * 997 * AmountNeeded

AmountNeeded == 527_899...97 ~= 527_899 * 10 ** 18;

// He need to pay additional 27_899 for this swap (fee + constant product math)

// Or he can call `mintFromReservoir` and `burn` functions, with the next arguments:

uint256 calculatedForFreeSwap ~= 456_540;

uint256 liqOut = vars.pair.mintWithReservoir(x + calculatedForFreeSwap, vars.swapper1); // 500_000 + ~456_540
vars.pair.burn(liqOut, vars.swapper1);

// As a result, he will have:

// tokenB = 500_009_147719996131821257 ~ 500_000
// tokenA = 499_990_852280003868178740 ~ 500_000

// He paid almost zero commission, abusing this feature.

// This works as well with `burnFromReservoir` function in another direction.
// mint (50/50) -> burnFromReservoir
// mintFromReservoir -> burnFromReservoir
```

Possible consequences:

1. Lp providers APY become APR (without any further manipulations and attacks).
2. New protocol or smart route can be created, using funds of your Lp providers, which will create a tx, where he can get the same fee (0.3%) to itself.
3. (If we assume, that we know rebase tx or approximate time/period) Lp providers, in all pools, that have reservoirs will constantly lose part of their income (even less than APR), as any rebase tx will be sandwiched for free. So arbitrators will receive a portion of the Lp's income and won't pay any fees. They could get net income from this attack.

Recommendation

We recommend adding fees calculation on both **mintFromReservoir** and **burnFromReservoir** functions. It should be calculated using a virtual swap result as if it is the usual one.

Client's comments

This is intended behavior, as it serves as an incentive to motivate pool-rebalancing before reservoirs get large. Made more explicit in the Readme: https://github.com/buttonwood-protocol/buttonswap-core/pull/100

HIGH-03	Potential loss of reservoir from arbitrage opportunities	Acknowledged

Description

The **mintWithReservoir** and **burnFromReservoir** functions utilize a moving average price, which isn't adaptive to immediate price fluctuations. This creates arbitrage opportunities that can lead to reservoir withdrawal, especially during periods of sudden market price changes. As a result, it can compromise the safety of the LP's funds.
Let's say we have reservoir of token A, so the balances are $A_t = A_p + A_r, B_t = B_p$, where index t stands for total balance, p stands for pool balance (without reservoir) and r stands for reservoir balance. Consider the bundle of **mint** and **burnFromReservoir**. Let $\alpha = \frac{L_u}{L_t}$, so a LP have to provide αA_t of token A and αB_t of token B to get L_u of liquidity tokens.
Let's calculate how many tokens A it can receive using **burnFromReservoir**:
$A_u = A_t^{'} \cdot \frac{L_u}{L_t^{'}} + B_t^{'} \cdot \frac{L_u}{L_t^{'}} \cdot \frac{1}{p_{ma}} = (1+\alpha)A_t \cdot \frac{L_u}{(1+\alpha)L_t} + (1+\alpha)B_t \frac{L_u}{(1+\alpha)L_t} \cdot \frac{1}{p_{ma}}$ Thus, $A_u = \alpha A_t + \alpha B_t \cdot \frac{1}{p_{ma}}$, so we can treat the bundle as swap $\alpha B_t = b$ of token B to $\alpha B_t \cdot \frac{1}{p_{ma}} = \frac{b}{p_{ma}}$ without any fee or slippage. It has limitation that $A_u \leq A_r \Leftrightarrow \frac{b}{B_t}A_t + \frac{b}{p_{ma}} \leq A_r \Leftrightarrow b \leq A_r(\frac{1}{p_{ma}} + \frac{A_t}{B_t})^{-1} \approx A_r(\frac{1}{p_{ma}} + \frac{1}{p_{ma}})^{-1} \Leftrightarrow \frac{b}{p_{ma}} \leq \frac{1}{2}A_r$ which means that it is possible to swap to almost the half of reservoir.
The problem arises primarily when there's a sudden change in the market price, and the moving average price (p_{ma}) still reflects the old price. For example, if the market price of token A relative to B rise by 10%, equating to $p_r = 1.1 \cdot p_{ma}$, which means that on other exchanges it is possible to swap a of token A to $a \cdot p_r$ of token B. This creates an arbitrage opportunity on the market and it is profitable to swap as many b of tokens B to $\frac{b}{p_{ma}}$ of tokens A. It could be said that the protocol (or LP providers) will lose $\frac{b}{p_{ma}} - \frac{b}{p_r} = \frac{1}{11} \cdot \frac{b}{p_{ma}} \approx \frac{1}{11} \cdot \frac{1}{2}A_r \approx 0.045A_r$ which is loss of approximately 4.5% of reservoir in case of price deviation by 10%.

Recommendation

The use of a moving average price, due to its inability to adapt quickly to price deviations, may result in substantial loss from the reservoir. It's recommended to avoid this pricing model for the **mintWithReservoir** and **burnFromReservoir** functions. Instead, consider employing the core concept from Uniswap V2 - defining an invariant that ensures a user's liquidity ($\frac{L_u}{L_t}\sqrt{A_t \cdot B_t}$) doesn't decrease during operations, but increases due to fees. While this approach does not entirely protect LPs from impermanent loss, it assures the safety of their funds and resilience to market changes.
For the **burnFromReservoir** function, suppose an LP wants to burn L_u of liquidity tokens and $\alpha = \frac{L_u}{L_t}$. In this case, the liquidity ($\sqrt{A_t \cdot B_t}$) should decrease to $(1-\alpha)\sqrt{A_t \cdot B_t}$. A token **A** should be returned in an amount a that satisfies the equation: $\sqrt{(A_t - a) \cdot B_t} = (1-\alpha)\sqrt{A_t \cdot B_t}$, which simplifies to $a = (1-(1-\alpha)^2)A_t$. Similar calculations can be applied to **mintWithResevoir**.

Client's comments

The key distinguishing feature of the Buttonswap Protocol over other AMMs is that it insulates the pools from immediate change in marginal price due to rebasing. The excess is preserved inside a reservoir rather than being forfeited to arbitrage opportunities. In order to address the capital inefficiency of growing reservoirs, the **mintWithReservoir** and **burnFromReservoir** functions allow reintroducing the reservoir assets into active liquidity as a benefit to the existing LPs. The **mintWithReservoir** and **burnFromReservoir** functions have three safeguards to mitigate loss of reservoir from arbitrage opportunities and ensure the safety of LP funds:

1. Moving Average Price: Keeps track of the time weighted average price ratio
2. Single-Sided Timelock: Blocks reservoir exchanges after large swaps
3. Swappable Reservoir Limit: Limits the size of reservoir exchanges

These safeguards are necessary because any interaction with a reservoir requires pricing the reservoir asset with respect to the other. Using the existing pool ratios creates the incentive for price manipulation to extract value from the LPs. Instead, the moving average price is utilized to estimate the "stable" market price while the single-sided time lock and swappable reservoir limit block interactions with the reservoir during excessive price variations. In short, rapid price deviations are intrinsically indifferentiable from pool price manipulations, and hence it becomes a preferable experience to protect LPs from value loss in the reservoir by restricting its exposure. Finally, maintaining a K constant as a liquidity invariant is incompatible with reservoirs. It relies on the assumption that A_t and B_t have the same relative change in balances. Since single-sided operations only increment the total balance of a single asset, changes to a user's liquidity ($\frac{L_u}{L_t}\sqrt{A_t \cdot B_t}$) would be skewed heavily to the asset with a higher balance.

MEDIUM-01	Imprecise value returned by _closestBound()	Fixed at 99c1e4

Description

The **_closestBound()** function, as currently implemented, exhibits certain issues that lead to imprecise value return:

1. The function always returns **poolALower** since it lacks an **else** statement, which limits its functionality.
2. The formula used in the function contains a division operation, introducing rounding errors.

Recommendation

It is recommended to implement the following changes:

1. Include an **else** statement in the function to ensure that **poolALower** is not always returned.
2. Modify the formula by multiplying it by 2. This adjustment helps reduce the rounding error introduced by the division operation.

MEDIUM-02	The paramSetter address can block single side mints and burns	Fixed at d3f649

Description

In the **_getSwappableReservoirLimit** function in the **ButtonswapPair** contract on line L346 there can be an underflow. It can happen in this cases:
There was a single side mint/burn in a pair, then the **paramSetter** address changes value of the **swappableReservoirGrowthWindow** variable to lower one. Then happens second single side mint/burn in the pair. In this case, there can be an underflow. Example:
At first, the **swappableReservoirGrowthWindow** variable was equal to **1000** seconds. Then, happens a single side mint/burn that requires half of the reserves for a swap. The variable **swappableReservoirLimitReachesMaxDeadline** in the pair will be equal to **block.timestamp + swappableReservoirGrowthWindow/2 = block.timestamp + 500**.
Then, the **paramSetter** address changes value of the **swappableReservoirGrowthWindow** variable to **200**.
After that happens the second single side mint/burn in a pair. In that transaction the contract will call the function **_getSwappableReservoirLimit** in which will be calculated the variable **progress**. The contract will try to subtract the value **_swappableReservoirLimitReachesMaxDeadline - block.timestamp** from the **swappableReservoirGrowthWindow** value and it will underflow.

Recommendation

When the contract sets the variable **swappableReservoirGrowthWindow** to a new value, adjust the **swappableReservoirLimitReachesMaxDeadline** variable:

```
function setSwappableReservoirGrowthWindow(...) external onlyFactory {
    uint32 oldSwappableReservoirGrowthWindow = swappableReservoirGrowthWindow;
    swappableReservoirGrowthWindow = _swappableReservoirGrowthWindow;

    uint256 oldSwappableReservoirLimitReachesMaxDeadline = swappableReservoirLimitReachesMaxDeadline;
    if (oldSwappableReservoirLimitReachesMaxDeadline > block.timestamp) {
        uint256 progress = oldSwappableReservoirGrowthWindow - (oldSwappableReservoirLimitReachesMaxDeadline -
block.timestamp);

        swappableReservoirLimitReachesMaxDeadline = _swappableReservoirGrowthWindow * progress /
oldSwappableReservoirGrowthWindow;
    }
}
```

MEDIUM-03	Fund locking vulnerability via isPausedSetter role	Fixed at d63805

Description

It is possible for **isPausedSetter** role to lock users' funds in the pair. This situation could occur when the **isPausedSetter** performs a **swap** operation that results in the balance of token A being reduced to one.
For instance, if the initial pool balances were (A_t, B_t), a swap would cause them to become $(1, A_t \cdot B_t \cdot 1.003)$, due to the 1.003 swap fee. Within the same transaction, if the **isPausedSetter** puts a pause on the pair, then **swap**, **mint**, **mintWithReservoir**, and **burnFromReservoir** operations will be blocked because the pair is paused.
Moreover, the **burn** operation will also become unavailable due to the following condition:

```
function burn(uint256 liquidityIn, address to)
        external
        lock
        sendOrRefundFee
        returns (uint256 amountOut0, uint256 amountOut1)
    {
        // ...

        (amountOut0, amountOut1) = PairMath.getDualSidedBurnOutputAmounts(_totalSupply, liquidityIn, total0, total1);

        if (amountOut0 == 0 || amountOut1 == 0) { // <-- will revert if amountOut0 is zero
            revert InsufficientLiquidityBurned();
        }
       // ...
    }
```

This condition will cause the function to revert if **amountOut0** becomes zero, which is indeed the case here due to the burn output amount calculations:

```
function getDualSidedBurnOutputAmounts(uint256 totalLiquidity, uint256 liquidityIn, uint256 totalA, uint256 totalB)
        internal
        pure
        returns (uint256 amountOutA, uint256 amountOutB)
    {
        amountOutA = (totalA * liquidityIn) / totalLiquidity; // <-- zero due to totalA being reduced to 1
        amountOutB = (totalB * liquidityIn) / totalLiquidity;
    }
```

Because **totalA** has been reduced to 1, **amountOutA** is zero, causing the **burn** function to revert and locking the users' funds in the pair.

Recommendation

We recommend using zero check for **liquidityIn** instead of **amountOut0** and **amountOut1**.

MEDIUM-04	Possibility of manipulation over movingAveragePrice0	Acknowledged

Description

In the function **movingAveragePrice0()**, **_movingAveragePrice0** is only updated in the first swap within the block. This means that there is an opportunity to create your own **currentPrice0** in every block (the price is calculated from the last swap in the previous block). If the market price goes higher or lower, we can handle the averagePrice in the same position, as the price hasn't changed, making it much easier than adjusting the price ourselves.
For example:

```
Price = y/x
We know that price will increase by up to +20% for any reason and there will be reservoir or there already is

1 block:
Attacker backruns rebase transaction and makes little swap ( for saving ratio )
* Some swaps by users *
Now price in pool - 1.2*y/x ( or close )
Moving averagePrice still the same (y/x)
attacker backruns last swap and rebalances pool by making swap for 20% of the liquidity
2 block:
attacker frontrun any swap and balance pool ( so he loses only 0.6% )
* Some swaps by users *
backrun of last swap
3 - (n-1) block:
same
n block:
when attacker have an opportunity to burnFromReservoir he withdraw tokens that swapped virtually for old price ( tokens in reservoir have a lower price )
```

Yes, in the normal case, the averagePrice won't update fast enough to reach 20%, but it will still grow. So we can create a higher discount for tokens in the reservoir. By manipulating this, the attacker can "steal" some tokens from the LP yield. There is no reason for users to frontrun the attacker. This manipulation can't affect the swap functionality. If price goes down we can make the same with **mintWithReservoir()** likewise.

Recommendation

We recommend fixing calculating averagePrice possible by considering not only first swap in block and how big amount of swap in terms of whole liquidity

Client's comments

We believe that the mitigation mechanisms would inhibit the ability for an exploit to significantly manipulate the movingAveragePrice. While there are edge-cases where volume-weighting would help mitigate exploit potential, overall volume-weighting diminishes the effectiveness of time-weighting (for example, somebody could use a flashbot bundle across 2 blocks with a large trade that has tangible volume-weighted impact despite neglible time-weighted impact). It's this reason we've opted to stick to time-weighted.

MEDIUM-05	Back-run/front-run of unpause	Acknowledged

Description

Since **movingAveragePrice0** and **PriceCumulative**s tied to **timeElapsed = blockTimestamp - blockTimestampLast**, the pause functionality can help to manipulate prices.
If the pause lasted longer than **24 hours**, then **movingAveragePrice0** becomes extra sensitive to manipulations.

```
uint256 currentPrice0 = uint256(UQ112x112.encode(pool1Last).uqdiv(pool0Last));
if (timeElapsed >= 24 hours)
    _movingAveragePrice0 = currentPrice0;
```

E.g.

1. Front-run pause to skew pool prices.
2. Back-run unpause to take advantage of manipulated prices.

Recommendation

We recommend pausing through private mempools.

Client's comments

Documentation updated in PR: https://github.com/buttonwood-protocol/buttonswap-core/pull/102

MEDIUM-06	DOS of mintWithReservoir and burnFromReservoir	Acknowledged

Description

In every swap after checking **K** invariant **singleSidedTimelockDeadline** is updated at Line 751. Based on calculations at Lines 320-327 **singleSidedTimelockDeadline** will be always greater or equal than **minTimelockDuration**. This means that the functionality of **mintWithReservoir** and **burnFromReservoir** will be blocked at least **minTimelockDuration**. Currently, default value for this param is 24 seconds (2 blocks). Malicious actor can call **swap()** function with minimal amounts every 2 blocks to extend the deadline. Another way of attack, this actor can take flashloan, make big swap and set lock to **maxTimelockDuration**.

Recommendation

It is recommended to check price difference before setting timelock with additional protocol param or using **maxVolatilityBps** and consider cases when timelock can be decreased based on decrease of price difference. In current implementation even with small price difference timelock will be set, also **maxVolatilityBps** is set to be 7% which is sensitive even to such changes.

Client's comments

This is expected behavior. DoS doesn't have any significant impact on the pair's ability to operate swaps. It only impacts reservoir-interactions which are non-critical. For a sustained DoS, the cost of the grief attack is non-trivial and has no economic incentive.

INFORMATIONAL-01	Gas optimizations: Loops Improvement	Fixed at 69dd25

Description

1. Iteration Incrementor: Consider using ++i instead of i++ to potentially optimize gas usage:
   - ButtonswapFactory.sol#L186
   - ButtonswapFactory.sol#L238
   - ButtonswapFactory.sol#L250
   - ButtonswapFactory.sol#L262
   - ButtonswapFactory.sol#L276
   - ButtonswapFactory.sol#L290
2. Array Length Caching: Rather than fetching the array length each time, it could be more efficient to cache this value:
   - ButtonswapFactory.sol#L186
   - ButtonswapFactory.sol#L238
   - ButtonswapFactory.sol#L250
   - ButtonswapFactory.sol#L262
   - ButtonswapFactory.sol#L276
   - ButtonswapFactory.sol#L290
3. Avoiding Redundant Defaults: Setting variables to their default values may not be necessary and could be avoided to optimize gas consumption:
   - ButtonswapFactory.sol#L186
   - ButtonswapFactory.sol#L238
   - ButtonswapFactory.sol#L250
   - ButtonswapFactory.sol#L262
   - ButtonswapFactory.sol#L276
   - ButtonswapFactory.sol#L290
4. Unchecked Math: Implement math operations that are resistant to errors - **unchecked**:
   - ButtonswapFactory.sol#L186
   - ButtonswapFactory.sol#L238
   - ButtonswapFactory.sol#L250
   - ButtonswapFactory.sol#L262
   - ButtonswapFactory.sol#L276
   - ButtonswapFactory.sol#L290

Recommendation

It is recommended to implement the suggested changes.

INFORMATIONAL-02	Code style improvements	Fixed at de6584

Description

The following code style improvements are recommended:

1. PairMath.sol#L119 - Remove redundant "@dev" keyword duplication.
2. PairMath.sol#L38 - Correct the misspelling of "uint112" as "unit112".
3. ButtonswapPair.sol#L450-L454 - Define a constant with a meaningful name for the value **24 hours**.

Recommendation

It is recommended to implement the suggested changes.

INFORMATIONAL-03	Gas optimizations: Caching Storage Variables	Fixed at fcc053

Description

Caching storage variables to memory or local variable:

- **minTimelockDuration** in ButtonswapPair.sol#L321, ButtonswapPair.sol#L323
- **maxTimelockDuration** in ButtonswapPair.sol#L323, ButtonswapPair.sol#L326
- **swappableReservoirGrowthWindow** in ButtonswapPair.sol#L348, ButtonswapPair.sol#L350
- In the function **mint** the global variables **pool1Last** and **pool0Last** can be read after write.

Recommendation

We recommend adding caching storage variables

INFORMATIONAL-04	Gas optimizations: sendOrRefundFee modifier	Fixed at 5dcd11

Description

sendOrRefundFee modifier make **_burn** or **_transfer** even if pair **LP balance == 0**. Also, **sendOrRefundFee** modifier used every call **ButtonswapPair::mint**, **ButtonswapPair::mintWithReservoir**, **ButtonswapPair::burn**, **ButtonswapPair::burnWithReservoir** functions. However, pair contract mint fee LP only in swap operation.

Recommendation

We recommend adding check **balanceOf[address(this)] > 0** for gas optimization

INFORMATIONAL-05	Pool lock funds if one of the tokens with zeroed balance	Fixed at ec04ee

Description

UniswapV2Pair denies **UniswapV2Pair::burn** if one of tokens has 0th balance UniswapV2Pair.sol#L146. Also, we have the assumption that pair balances in the rebasing token can be decreased to 0 in critical cases. For example normal negative rebase on the amount of liquidity in the pool, hack rebasing token and still underlying liquidity, a rebasing token owner manipulates balances own token, CBDC authority burns one of the tokens in the pool because hacker adds a lot of liquidities, etc If one of the tokens has 0th balance, **ButtonswapPair::mint**, **ButtonswapPair::mintWithReservoir**, **ButtonswapPair::burn**, **ButtonswapPair::burnWithReservoir**, **ButtonswapPair::swap** always reverts. However, LP providers want to return at least part of their funds in the opposite token.

Recommendation

We recommend extending **ButtonswapPair::burn** function and allowing burn LP even one of the amount outputs is 0.

```
function burn(uint256 liquidityIn, address to, bool allowZeroOutput)
    external
    lock
    sendOrRefundFee
    returns (uint256 amountOut0, uint256 amountOut1)
{
   ...
   if (!allowZeroOutput && (amountOut0 == 0 || amountOut1 == 0)) {
      revert InsufficientLiquidityBurned();
   }
   ...
}
```

INFORMATIONAL-06	Unbounded configuration parameters	Fixed at d3f649

Description

The configuration parameters are not limited giving the roles ability to break the core logic.
E.g.

1. **minTimelockDuration > maxTimelockDuration**
2. **maxVolatilityBps > BPS or maxVolatilityBps = 0**
3. **maxSwappableReservoirLimitBps > BPS**
4. **swappableReservoirGrowthWindow = 0**

Recommendation

We recommend introducing constant variables limiting configuration parameters.

Client's comments

> Addressed in PRs: https://github.com/buttonwood-protocol/buttonswap-core/pull/101, https://github.com/buttonwood-protocol/buttonswap-core/pull/103

INFORMATIONAL-07	ERC20 code duplication	Fixed at 3fe9f4

Description

The **transferFrom** function uses implemented functionality of **_approve** without calling it.

Recommendation

We recommend changing duplicated code to the internal **_approve** function call.

INFORMATIONAL-08	No events in setters	Fixed at edbd1b

Description

In the contracts **ButtonswapPair** and **ButtonswapFactory** in all admin's setters there are no events.

Recommendation

Consider adding appropriate events into the admin's setters.

INFORMATIONAL-09	Little gas optimizations	Acknowledged

Description

1.Can be unchecked because **totalSupply** will revert first.
2.Can be unchecked because **balanceOf[from]** will revert first.
3.Can use **total0 lb.pool1 = (lb.pool0 * _pool1Last) / _pool0Last;** instead of **lb.pool0**. Same this **total1** and **lb.pool1**
4.Check can be optimized by **if (pool0New*pool1New == 0)**

Recommendation

We recommend exploring the possibility of optimization.

Client's comments

> We prefer the existing code-clarity over the minor gas savings.

INFORMATIONAL-10	Possible manipulation of reservoirs by directly transferring tokens	Acknowledged

Description

The function **mintWithReservoir()** only specifies the **amountIn** parameter but not the token being used to mint. Let us say the user wants to mint using **token0**. A malicious user can directly transfer **token1** such that the **reservoir1** becomes non zero. Then the contract will use the **token1** approval of **msg.sender** to mint using **token1** instead of **token0**.

Recommendation

We recommend to ensure such a scenario is not possible in the periphery contracts.

Client's comments

> These protections already exist in the router contracts.

INFORMATIONAL-11	movingAveragePrice0Last gas optimization	Acknowledged

Description

It is possible to save gas if **movingAveragePrice0Last** is only updated if it is different than **_movingAveragePrice0** at the line ButtonswapPair.sol#L748.

Recommendation

We recommend to update **movingAveragePrice0Last** only if it changes.

Client's comments

> The proposed change adds a warm SLOAD from movingAveragePrice0Last every swap, costing 100 gas. It can at times save a warm same-value SSTORE to movingAveragePrice0Last on some swaps, saving 100 gas. Due to movingAveragePrice0 being calculated by interpolating values based on current time, it is very often going to be different from the last stored value (the most likely exception is when two swaps occur in the same block). On balance we believe always updating it results in lower average gas consumption rather than only updating when the value has changed.

INFORMATIONAL-12	Minting and burning from reservoir don't have a slippage protection	Acknowledged

Description

The amount of tokens minted or burned is determined by **movingAveragePrice0**, which is updated on every swap. Every swap extends **singleSidedTimelockDeadline** lock by at least 24 seconds (as defined in **MIN_TIMELOCK_DURATION**). Under heavy network load, it is possible that transaction attempting to mint or burn from only one of reservoirs isn't included in the block for a while and is front-runned by a swap, which will change the amount of tokens minted/burned by one-sided operation.

Recommendation

It is important to include slippage protection and deadline timestamp for one-sided operations inside the Router contract.

Client's comments

> These protections already exist in the router contracts.

INFORMATIONAL-13	Risk Associated with isPausedSetter Role	Acknowledged

Description

The **isPausedSetter** role holds significant power with potentially disruptive effects on the system. This role has the ability to pause essential functions such as **swap**, **mint**, **mintWithReservoir**, and **burnFromReservoir**.

```
// The following functions can be paused by the `isPausedSetter` role:

function swap(...) external checkPaused {...}

function mint(...) external checkPaused {...}

function mintWithReservoir(...) external checkPaused {...}

function burnFromReservoir(...) external checkPaused {...}
```

The ability to pause **swap** is particularly concerning. Without the ability to swap, the token ratio in the pool might deviate from the market ratio. This discrepancy creates an arbitrage opportunity, which would likely lead liquidity providers to withdraw their liquidity from the pool in order to avoid losses, resulting in significant liquidity reduction.

Recommendation

It is recommended to implement additional safeguards around the **isPausedSetter** role, e.g. using multisig.

Client's comments

Will take appropriate precautions when handling roles.

6. Appendix A. Linter

Error/max-states-count

- ButtonswapERC20.sol:6 – Contract has 4 states declarations but allowed no more than 3
- ButtonswapFactory.sol:8 – Contract has 15 states declarations but allowed no more than 3
- ButtonswapPair.sol:13 – Contract has 15 states declarations but allowed no more than 3

Error/const-name-snakecase

- ButtonswapERC20.sol:10 – Constant name must be in capitalized SNAKE_CASE
- ButtonswapERC20.sol:15 – Constant name must be in capitalized SNAKE_CASE
- ButtonswapERC20.sol:20 – Constant name must be in capitalized SNAKE_CASE

Error/var-name-mixedcase

- ButtonswapERC20.sol:40 – Variable name must be in mixedCase
- interfaces/IButtonswapERC20/IButtonswapERC20.sol:91 – Variable name must be in mixedCase
- interfaces/IButtonswapERC20/IButtonswapERC20.sol:97 – Variable name must be in mixedCase
- interfaces/IButtonswapPair/IButtonswapPair.sol:18 – Variable name must be in mixedCase

Error/ordering

- ButtonswapERC20.sol:118 – Function order is incorrect, internal function can not go after private function (line 105)
- ButtonswapFactory.sol:107 – Function order is incorrect, external function can not go after external view function (line 100)
- ButtonswapPair.sol:267 – Function order is incorrect, internal view function can not go after internal pure function (line 250)
- interfaces/IButtonswapERC20/IButtonswapERC20.sol:32 – Function order is incorrect, external view function can not go after external pure function (line 26)
- interfaces/IButtonswapFactory/IButtonswapFactory.sol:65 – Function order is incorrect, external function can not go after external view function (line 56)
- interfaces/IButtonswapPair/IButtonswapPair.sol:24 – Function order is incorrect, external view function can not go after external pure function (line 18)

Error/not-rely-on-time

- ButtonswapERC20.sol:159 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:149 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:226 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:328 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:345 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:348 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:395 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:401 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:441 – Avoid making time-based decisions in your business logic
- ButtonswapPair.sol:485 – Avoid making time-based decisions in your business logic

Error/private-vars-leading-underscore

- ButtonswapFactory.sol:29 – 'lastToken0' should start with _
- ButtonswapFactory.sol:31 – 'lastToken1' should start with _
- ButtonswapPair.sol:38 – 'BPS' should start with _
- ButtonswapPair.sol:84 – 'pool0Last' should start with _
- ButtonswapPair.sol:90 – 'pool1Last' should start with _
- ButtonswapPair.sol:95 – 'blockTimestampLast' should start with _
- ButtonswapPair.sol:110 – 'movingAveragePrice0Last' should start with _
- ButtonswapPair.sol:126 – 'isPaused' should start with _
- ButtonswapPair.sol:131 – 'unlocked' should start with _
- libraries/UQ112x112.sol:10 – 'Q112' should start with _

Error/no-empty-blocks

- ButtonswapPair.sol:274 – Code contains empty blocks
- ButtonswapPair.sol:276 – Code contains empty blocks

Error/function-max-lines

- ButtonswapPair.sol:506 – Function body contains 76 lines but allowed no more than 40 lines
- ButtonswapPair.sol:612 – Function body contains 59 lines but allowed no more than 40 lines
- ButtonswapPair.sol:677 – Function body contains 79 lines but allowed no more than 40 lines
- libraries/Math.sol:13 – Function body contains 62 lines but allowed no more than 40 lines
- libraries/Math.sol:83 – Function body contains 75 lines but allowed no more than 40 lines

Error/code-complexity

- ButtonswapPair.sol:506 – Function has cyclomatic complexity 11 but allowed no more than 5
- ButtonswapPair.sol:612 – Function has cyclomatic complexity 9 but allowed no more than 5
- ButtonswapPair.sol:677 – Function has cyclomatic complexity 13 but allowed no more than 5

Error/func-name-mixedcase

- interfaces/IButtonswapERC20/IButtonswapERC20.sol:91 – Function name must be in mixedCase
- interfaces/IButtonswapERC20/IButtonswapERC20.sol:97 – Function name must be in mixedCase
- interfaces/IButtonswapPair/IButtonswapPair.sol:18 – Function name must be in mixedCase

Error/state-visibility

- libraries/UQ112x112.sol:10 – Explicitly mark visibility of state

7. Appendix B. Slither

Informational/High/assembly

Math.sqrt(uint256) uses assembly

- INLINE ASM

ButtonswapFactory.createPair(address,address) uses assembly

- INLINE ASM

Math.mulDiv(uint256,uint256,uint256) uses assembly

- INLINE ASM
- INLINE ASM
- INLINE ASM

ButtonswapERC20.constructor() uses assembly

- INLINE ASM

Informational/High/cyclomatic-complexity

ButtonswapPair.swap(uint256,uint256,uint256,uint256,address) has a high cyclomatic complexity (13).

Informational/High/naming-convention

Parameter ButtonswapPair.setMinTimelockDuration(uint32)._minTimelockDuration is not in mixedCase

Parameter ButtonswapFactory.setFeeToSetter(address)._feeToSetter is not in mixedCase

Parameter ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxSwappableReservoirLimitBps is not in mixedCase

Parameter ButtonswapFactory.setFeeTo(address)._feeTo is not in mixedCase

Parameter ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultSwappableReservoirGrowthWindow is not in mixedCase

Parameter ButtonswapFactory.setIsCreationRestricted(bool)._isCreationRestricted is not in mixedCase

Function IButtonswapERC20.DOMAIN_SEPARATOR() is not in mixedCase

Parameter ButtonswapFactory.setIsCreationRestrictedSetter(address)._isCreationRestrictedSetter is not in mixedCase

Parameter ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxVolatilityBps is not in mixedCase

Parameter ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMinTimelockDuration is not in mixedCase

Parameter ButtonswapPair.setMaxTimelockDuration(uint32)._maxTimelockDuration is not in mixedCase

Function IButtonswapPair.MINIMUM_LIQUIDITY() is not in mixedCase

Parameter ButtonswapPair.setMaxSwappableReservoirLimitBps(uint16)._maxSwappableReservoirLimitBps is not in mixedCase

Parameter ButtonswapFactory.setIsPausedSetter(address)._isPausedSetter is not in mixedCase

Parameter ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxTimelockDuration is not in mixedCase

Function IButtonswapERC20.PERMIT_TYPEHASH() is not in mixedCase

Parameter ButtonswapFactory.setParamSetter(address)._paramSetter is not in mixedCase

Variable ButtonswapERC20.DOMAIN_SEPARATOR is not in mixedCase

Parameter ButtonswapPair.setSwappableReservoirGrowthWindow(uint32)._swappableReservoirGrowthWindow is not in mixedCase

Parameter ButtonswapPair.setMaxVolatilityBps(uint16)._maxVolatilityBps is not in mixedCase

Informational/High/solc-version

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

solc-0.8.19 is not recommended for deployment

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Pragma version^0.8.13 allows old versions

Informational/Medium/similar-names

Variable ButtonswapPair.getLiquidityBalances()._reservoir0 is too similar to IButtonswapPair.getLiquidityBalances()._reservoir1

Variable IButtonswapPair.getLiquidityBalances()._reservoir0 is too similar to IButtonswapPair.getLiquidityBalances()._reservoir1

Variable PairMath.getSingleSidedBurnOutputAmountA(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountA is too similar to PairMath.getSingleSidedMintLiquidityOutAmountA(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountB

Variable IButtonswapPair.price0CumulativeLast().price0CumulativeLast is too similar to ButtonswapPair.price1CumulativeLast

Variable ButtonswapPair.mintWithReservoir(uint256,address).swappedReservoirAmount0 is too similar to ButtonswapPair.mintWithReservoir(uint256,address).swappedReservoirAmount1

Variable IButtonswapFactory.setMaxTimelockDuration(address[],uint32).newMaxTimelockDuration is too similar to ButtonswapFactory.setMinTimelockDuration(address[],uint32).newMinTimelockDuration

Variable ButtonswapPair.mintWithReservoir(uint256,address).swappedReservoirAmount0 is too similar to ButtonswapPair.burnFromReservoir(uint256,address).swappedReservoirAmount1

Variable IButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxTimelockDuration is too similar to IButtonswapFactory.defaultMinTimelockDuration()._defaultMinTimelockDuration

Variable IButtonswapFactory.defaultMaxTimelockDuration()._defaultMaxTimelockDuration is too similar to IButtonswapFactory.defaultMinTimelockDuration()._defaultMinTimelockDuration

Variable PairMath.getSingleSidedBurnOutputAmountA(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountA is too similar to PairMath.getSingleSidedBurnOutputAmountB(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountB

Variable PairMath.getSingleSidedMintLiquidityOutAmountA(uint256,uint256,uint256,uint256,uint256).mintAmountA is too similar to PairMath.getSingleSidedMintLiquidityOutAmountB(uint256,uint256,uint256,uint256,uint256).mintAmountB

Variable ButtonswapFactory.setMaxTimelockDuration(address[],uint32).newMaxTimelockDuration is too similar to IButtonswapFactory.setMinTimelockDuration(address[],uint32).newMinTimelockDuration

Variable ButtonswapPair.burnFromReservoir(uint256,address).swappedReservoirAmount0 is too similar to ButtonswapPair.mintWithReservoir(uint256,address).swappedReservoirAmount1

Variable PairMath.getSingleSidedMintLiquidityOutAmountB(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountA is too similar to PairMath.getSingleSidedMintLiquidityOutAmountA(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountB

Variable IButtonswapPair.price0CumulativeLast().price0CumulativeLast is too similar to IButtonswapPair.price1CumulativeLast().price1CumulativeLast

Variable IButtonswapFactory.setMaxTimelockDuration(address[],uint32).newMaxTimelockDuration is too similar to IButtonswapFactory.setMinTimelockDuration(address[],uint32).newMinTimelockDuration

Variable IButtonswapPair.getLiquidityBalances()._reservoir0 is too similar to ButtonswapPair.getLiquidityBalances()._reservoir1

Variable ButtonswapPair.price0CumulativeLast is too similar to IButtonswapPair.price1CumulativeLast().price1CumulativeLast

Variable ButtonswapPair.burnFromReservoir(uint256,address).swappedReservoirAmount0 is too similar to ButtonswapPair.burnFromReservoir(uint256,address).swappedReservoirAmount1

Variable ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxTimelockDuration is too similar to ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMinTimelockDuration

Variable IButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxTimelockDuration is too similar to ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMinTimelockDuration

Variable IButtonswapFactory.defaultMaxTimelockDuration()._defaultMaxTimelockDuration is too similar to IButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMinTimelockDuration

Variable ButtonswapPair.price0CumulativeLast is too similar to ButtonswapPair.price1CumulativeLast

Variable IButtonswapFactory.defaultMaxTimelockDuration()._defaultMaxTimelockDuration is too similar to ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMinTimelockDuration

Variable PairMath.getSingleSidedMintLiquidityOutAmountA(uint256,uint256,uint256,uint256,uint256).tokenAToSwap is too similar to PairMath.getSingleSidedMintLiquidityOutAmountB(uint256,uint256,uint256,uint256,uint256).tokenBToSwap

Variable ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxTimelockDuration is too similar to IButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMinTimelockDuration

Variable ButtonswapPair.swap(uint256,uint256,uint256,uint256,address).pool0NewAdjusted is too similar to ButtonswapPair.swap(uint256,uint256,uint256,uint256,address).pool1NewAdjusted

Variable ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxTimelockDuration is too similar to IButtonswapFactory.defaultMinTimelockDuration()._defaultMinTimelockDuration

Variable ButtonswapFactory.defaultMaxTimelockDuration is too similar to ButtonswapFactory.defaultMinTimelockDuration

Variable ButtonswapPair.burnFromReservoir(uint256,address).swappableReservoirLimit_scope_0 is too similar to ButtonswapPair.mintWithReservoir(uint256,address).swappableReservoirLimit_scope_1

Variable PairMath.getSingleSidedMintLiquidityOutAmountB(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountA is too similar to PairMath.getSingleSidedBurnOutputAmountB(uint256,uint256,uint256,uint256,uint256).swappedReservoirAmountB

Variable IButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMaxTimelockDuration is too similar to IButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32)._defaultMinTimelockDuration

Variable PairMath.getSingleSidedMintLiquidityOutAmountA(uint256,uint256,uint256,uint256,uint256).tokenARemaining is too similar to PairMath.getSingleSidedMintLiquidityOutAmountB(uint256,uint256,uint256,uint256,uint256).tokenBRemaining

Variable ButtonswapPair.getLiquidityBalances()._reservoir0 is too similar to ButtonswapPair.getLiquidityBalances()._reservoir1

Variable ButtonswapFactory.setMaxTimelockDuration(address[],uint32).newMaxTimelockDuration is too similar to ButtonswapFactory.setMinTimelockDuration(address[],uint32).newMinTimelockDuration

Informational/Medium/too-many-digits

Math.sqrt(uint256) uses literals with too many digits:

- ! y_sqrt_asm_0 < 0x1000000

Math.sqrt(uint256) uses literals with too many digits:

- ! y_sqrt_asm_0 < 0x10000000000

Math.sqrt(uint256) uses literals with too many digits:

- ! y_sqrt_asm_0 < 0x100000000000000000000000000000000

Math.sqrt(uint256) uses literals with too many digits:

- ! y_sqrt_asm_0 < 0x1000000000000000000

ButtonswapFactory.createPair(address,address) uses literals with too many digits:

- bytecode = type()(ButtonswapPair).creationCode

Low/High/shadowing-local

IButtonswapPair.price1CumulativeLast().price1CumulativeLast shadows:

- IButtonswapPair.price1CumulativeLast() (function)

IButtonswapPair.swappableReservoirLimitReachesMaxDeadline().swappableReservoirLimitReachesMaxDeadline shadows:

- IButtonswapPair.swappableReservoirLimitReachesMaxDeadline() (function)

IButtonswapERC20.name().name shadows:

- IButtonswapERC20.name() (function)

IButtonswapERC20.symbol().symbol shadows:

- IButtonswapERC20.symbol() (function)

IButtonswapERC20.decimals().decimals shadows:

- IButtonswapERC20.decimals() (function)

IButtonswapERC20.PERMIT_TYPEHASH().PERMIT_TYPEHASH shadows:

- IButtonswapERC20.PERMIT_TYPEHASH() (function)

IButtonswapPair.factory().factory shadows:

- IButtonswapPair.factory() (function)

IButtonswapPair.token1().token1 shadows:

- IButtonswapPair.token1() (function)

IButtonswapERC20.DOMAIN_SEPARATOR().DOMAIN_SEPARATOR shadows:

- IButtonswapERC20.DOMAIN_SEPARATOR() (function)

IButtonswapERC20.totalSupply().totalSupply shadows:

- IButtonswapERC20.totalSupply() (function)

IButtonswapPair.MINIMUM_LIQUIDITY().MINIMUM_LIQUIDITY shadows:

- IButtonswapPair.MINIMUM_LIQUIDITY() (function)

IButtonswapERC20.allowance(address,address).allowance shadows:

- IButtonswapERC20.allowance(address,address) (function)

IButtonswapPair.singleSidedTimelockDeadline().singleSidedTimelockDeadline shadows:

- IButtonswapPair.singleSidedTimelockDeadline() (function)

IButtonswapPair.price0CumulativeLast().price0CumulativeLast shadows:

- IButtonswapPair.price0CumulativeLast() (function)

IButtonswapPair.token0().token0 shadows:

- IButtonswapPair.token0() (function)

Low/Medium/calls-loop

ButtonswapFactory.setMaxVolatilityBps(address[],uint16) has external calls inside a loop: IButtonswapPair(pairs[i]).setMaxVolatilityBps(newMaxVolatilityBps)

ButtonswapFactory.setMinTimelockDuration(address[],uint32) has external calls inside a loop: IButtonswapPair(pairs[i]).setMinTimelockDuration(newMinTimelockDuration)

ButtonswapFactory.setMaxTimelockDuration(address[],uint32) has external calls inside a loop: IButtonswapPair(pairs[i]).setMaxTimelockDuration(newMaxTimelockDuration)

ButtonswapFactory.setMaxSwappableReservoirLimitBps(address[],uint16) has external calls inside a loop: IButtonswapPair(pairs[i]).setMaxSwappableReservoirLimitBps(newMaxSwappableReservoirLimitBps)

ButtonswapFactory.setSwappableReservoirGrowthWindow(address[],uint32) has external calls inside a loop: IButtonswapPair(pairs[i]).setSwappableReservoirGrowthWindow(newSwappableReservoirGrowthWindow)

ButtonswapFactory.setIsPaused(address[],bool) has external calls inside a loop: IButtonswapPair(pairs[i]).setIsPaused(isPausedNew)

Low/Medium/events-maths

ButtonswapPair.setMaxTimelockDuration(uint32) should emit an event for:

- maxTimelockDuration = _maxTimelockDuration

ButtonswapPair.setMinTimelockDuration(uint32) should emit an event for:

- minTimelockDuration = _minTimelockDuration

ButtonswapPair.setMaxSwappableReservoirLimitBps(uint16) should emit an event for:

- maxSwappableReservoirLimitBps = _maxSwappableReservoirLimitBps

ButtonswapPair.setSwappableReservoirGrowthWindow(uint32) should emit an event for:

- swappableReservoirGrowthWindow = _swappableReservoirGrowthWindow

ButtonswapPair.setMaxVolatilityBps(uint16) should emit an event for:

- maxVolatilityBps = _maxVolatilityBps

ButtonswapFactory.setDefaultParameters(uint16,uint32,uint32,uint16,uint32) should emit an event for:

- defaultMaxVolatilityBps = _defaultMaxVolatilityBps
- defaultMinTimelockDuration = _defaultMinTimelockDuration
- defaultMaxTimelockDuration = _defaultMaxTimelockDuration
- defaultMaxSwappableReservoirLimitBps = _defaultMaxSwappableReservoirLimitBps
- defaultSwappableReservoirGrowthWindow = _defaultSwappableReservoirGrowthWindow

Low/Medium/missing-zero-check

ButtonswapFactory.setIsPausedSetter(address)._isPausedSetter lacks a zero-check on :

- isPausedSetter = _isPausedSetter

ButtonswapFactory.setFeeToSetter(address)._feeToSetter lacks a zero-check on :

- feeToSetter = _feeToSetter

ButtonswapFactory.constructor(address,address,address,address)._paramSetter lacks a zero-check on :

- paramSetter = _paramSetter

ButtonswapFactory.setParamSetter(address)._paramSetter lacks a zero-check on :

- paramSetter = _paramSetter

ButtonswapFactory.constructor(address,address,address,address)._isPausedSetter lacks a zero-check on :

- isPausedSetter = _isPausedSetter

ButtonswapFactory.setFeeTo(address)._feeTo lacks a zero-check on :

- feeTo = _feeTo

ButtonswapFactory.constructor(address,address,address,address)._isCreationRestrictedSetter lacks a zero-check on :

- isCreationRestrictedSetter = _isCreationRestrictedSetter

ButtonswapFactory.setIsCreationRestrictedSetter(address)._isCreationRestrictedSetter lacks a zero-check on :

- isCreationRestrictedSetter = _isCreationRestrictedSetter

ButtonswapFactory.constructor(address,address,address,address)._feeToSetter lacks a zero-check on :

- feeToSetter = _feeToSetter

Low/Medium/reentrancy-benign

Reentrancy in ButtonswapPair.mint(uint256,uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn0)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn1) State variables written after the call(s):
- blockTimestampLast = uint32(block.timestamp % 2 ** 32)
- movingAveragePrice0Last = uint256(UQ112x112.encode(pool1Last).uqdiv(pool0Last))
- pool0Last = uint112(amountIn0)
- pool1Last = uint112(amountIn1)

Reentrancy in ButtonswapPair.mintWithReservoir(uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn) State variables written after the call(s):
- _updateSwappableReservoirDeadline(lb.pool1,swappedReservoirAmount1)
- swappableReservoirLimitReachesMaxDeadline = uint120(_swappableReservoirLimitReachesMaxDeadline + delay)
- swappableReservoirLimitReachesMaxDeadline = uint120(block.timestamp + delay)

Reentrancy in ButtonswapPair.mintWithReservoir(uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn) State variables written after the call(s):
- _updateSwappableReservoirDeadline(lb.pool0,swappedReservoirAmount0)
- swappableReservoirLimitReachesMaxDeadline = uint120(_swappableReservoirLimitReachesMaxDeadline + delay)
- swappableReservoirLimitReachesMaxDeadline = uint120(block.timestamp + delay)

Reentrancy in ButtonswapPair.swap(uint256,uint256,uint256,uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn0)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn1)
- SafeERC20.safeTransfer(IERC20(token0),to,amountOut0)
- SafeERC20.safeTransfer(IERC20(token1),to,amountOut1) State variables written after the call(s):
- _mintFee(lb.pool0,lb.pool1,pool0New,pool1New)
- balanceOf[to] = balanceOf[to] + value
- _updatePriceCumulative(lb.pool0,lb.pool1)
- blockTimestampLast = blockTimestamp
- movingAveragePrice0Last = _movingAveragePrice0
- _updatePriceCumulative(lb.pool0,lb.pool1)
- price0CumulativeLast += ((pool1 << 112) * timeElapsed) / _pool0
- _updatePriceCumulative(lb.pool0,lb.pool1)
- price1CumulativeLast += ((pool0 << 112) * timeElapsed) / _pool1
- _updateSingleSidedTimelock(_movingAveragePrice0,uint112(pool0New),uint112(pool1New))
- singleSidedTimelockDeadline = timelockDeadline
- _mintFee(lb.pool0,lb.pool1,pool0New,pool1New)
- totalSupply = totalSupply + value

Low/Medium/reentrancy-events

Reentrancy in ButtonswapPair.swap(uint256,uint256,uint256,uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn0)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn1)
- SafeERC20.safeTransfer(IERC20(token0),to,amountOut0)
- SafeERC20.safeTransfer(IERC20(token1),to,amountOut1) Event emitted after the call(s):
- Swap(msg.sender,amountIn0,amountIn1,amountOut0,amountOut1,to)
- Transfer(address(0),to,value)
- _mintFee(lb.pool0,lb.pool1,pool0New,pool1New)

Reentrancy in ButtonswapPair.mint(uint256,uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn0)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn1) Event emitted after the call(s):
- Mint(msg.sender,amountIn0,amountIn1,liquidityOut,to)
- Transfer(address(0),to,value)
- _mint(to,liquidityOut)
- Transfer(address(0),to,value)
- _mint(address(0),MINIMUM_LIQUIDITY)

Reentrancy in ButtonswapPair.mintWithReservoir(uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn) Event emitted after the call(s):
- Mint(msg.sender,amountIn,0,liquidityOut,to)
- Mint(msg.sender,0,amountIn,liquidityOut,to)
- Transfer(address(0),to,value)
- _mint(to,liquidityOut)

Reentrancy in ButtonswapPair.burn(uint256,address): External calls:

- SafeERC20.safeTransfer(IERC20(token0),to,amountOut0)
- SafeERC20.safeTransfer(IERC20(token1),to,amountOut1) Event emitted after the call(s):
- Burn(msg.sender,liquidityIn,amountOut0,amountOut1,to)

Reentrancy in ButtonswapPair.burnFromReservoir(uint256,address): External calls:

- SafeERC20.safeTransfer(IERC20(token0),to,amountOut0)
- SafeERC20.safeTransfer(IERC20(token1),to,amountOut1) Event emitted after the call(s):
- Burn(msg.sender,liquidityIn,amountOut0,amountOut1,to)

Low/Medium/timestamp

ButtonswapERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) uses timestamp for comparisons Dangerous comparisons:

- block.timestamp > deadline

ButtonswapPair._updatePriceCumulative(uint256,uint256) uses timestamp for comparisons Dangerous comparisons:

- timeElapsed > 0 && pool0 != 0 && pool1 != 0

ButtonswapPair._updateSingleSidedTimelock(uint256,uint112,uint112) uses timestamp for comparisons Dangerous comparisons:

- timelockDeadline > singleSidedTimelockDeadline

ButtonswapPair._updateSwappableReservoirDeadline(uint256,uint256) uses timestamp for comparisons Dangerous comparisons:

- _swappableReservoirLimitReachesMaxDeadline > block.timestamp

ButtonswapPair._getSwappableReservoirLimit(uint256) uses timestamp for comparisons Dangerous comparisons:

- _swappableReservoirLimitReachesMaxDeadline > block.timestamp

ButtonswapPair.movingAveragePrice0() uses timestamp for comparisons Dangerous comparisons:

- timeElapsed == 0
- timeElapsed >= 86400

Medium/High/incorrect-equality

ButtonswapPair.mint(uint256,uint256,address) uses a dangerous strict equality:

- liquidityOut == 0

ButtonswapPair._getLiquidityBalances(uint256,uint256) uses a dangerous strict equality:

- total0 == 0 || total1 == 0

ButtonswapPair.mintWithReservoir(uint256,address) uses a dangerous strict equality:

- liquidityOut == 0

ButtonswapPair.swap(uint256,uint256,uint256,uint256,address) uses a dangerous strict equality:

- pool0New == 0 || pool1New == 0

ButtonswapPair.movingAveragePrice0() uses a dangerous strict equality:

- timeElapsed == 0

ButtonswapPair.burn(uint256,address) uses a dangerous strict equality:

- amountOut0 == 0 || amountOut1 == 0

ButtonswapPair.swap(uint256,uint256,uint256,uint256,address) uses a dangerous strict equality:

- amountIn0 == 0 && amountIn1 == 0

ButtonswapPair._getLiquidityBalances(uint256,uint256) uses a dangerous strict equality:

- _pool0Last == 0 || _pool1Last == 0

Medium/High/write-after-write

ButtonswapFactory.lastToken1 is written in both lastToken1 = token1 lastToken1 = address(0)

ButtonswapFactory.lastToken0 is written in both lastToken0 = token0 lastToken0 = address(0)

Medium/Medium/divide-before-multiply

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- denominator = denominator / twos
- inverse *= 2 - denominator * inverse

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- prod0 = prod0 / twos
- result = prod0 * inverse

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- denominator = denominator / twos
- inverse *= 2 - denominator * inverse

PairMath.getSingleSidedMintLiquidityOutAmountA(uint256,uint256,uint256,uint256,uint256) performs a multiplication on the result of a division:

- tokenAToSwap = (mintAmountA * totalB) / (Math.mulDiv(movingAveragePriceA,(totalA + mintAmountA),2 ** 112) + totalB)
- swappedReservoirAmountB = (tokenAToSwap * movingAveragePriceA) / 2 ** 112

PairMath.getSingleSidedMintLiquidityOutAmountB(uint256,uint256,uint256,uint256,uint256) performs a multiplication on the result of a division:

- tokenBToSwap = (mintAmountB * totalA) / (((2 ** 112 * (totalB + mintAmountB)) / movingAveragePriceA) + totalA)
- swappedReservoirAmountA = (tokenBToSwap * (2 ** 112)) / movingAveragePriceA

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- denominator = denominator / twos
- inverse *= 2 - denominator * inverse

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- denominator = denominator / twos
- inverse *= 2 - denominator * inverse

ButtonswapPair._getSwappableReservoirLimit(uint256) performs a multiplication on the result of a division:

- maxSwappableReservoirLimit = (poolA * maxSwappableReservoirLimitBps) / BPS
- swappableReservoir = (maxSwappableReservoirLimit * progress) / swappableReservoirGrowthWindow

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- denominator = denominator / twos
- inverse = (3 * denominator) ^ 2

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- denominator = denominator / twos
- inverse *= 2 - denominator * inverse

Math.mulDiv(uint256,uint256,uint256) performs a multiplication on the result of a division:

- denominator = denominator / twos
- inverse *= 2 - denominator * inverse

Medium/Medium/reentrancy-no-eth

Reentrancy in ButtonswapPair.swap(uint256,uint256,uint256,uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn0)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn1)
- SafeERC20.safeTransfer(IERC20(token0),to,amountOut0)
- SafeERC20.safeTransfer(IERC20(token1),to,amountOut1) State variables written after the call(s):
- pool0Last = uint112(pool0New) ButtonswapPair.pool0Last can be used in cross function reentrancies:
- ButtonswapPair._getLiquidityBalances(uint256,uint256)
- ButtonswapPair.mint(uint256,uint256,address)
- ButtonswapPair.movingAveragePrice0()
- ButtonswapPair.swap(uint256,uint256,uint256,uint256,address)
- pool1Last = uint112(pool1New) ButtonswapPair.pool1Last can be used in cross function reentrancies:
- ButtonswapPair._getLiquidityBalances(uint256,uint256)

- ButtonswapPair.mint(uint256,uint256,address)
- ButtonswapPair.movingAveragePrice0()
- ButtonswapPair.swap(uint256,uint256,uint256,uint256,address)

Reentrancy in ButtonswapPair.mintWithReservoir(uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn) State variables written after the call(s):
- _mint(to,liquidityOut)
- balanceOf[to] = balanceOf[to] + value ButtonswapERC20.balanceOf can be used in cross function reentrancies:
- ButtonswapERC20._burn(address,uint256)
- ButtonswapERC20._mint(address,uint256)
- ButtonswapERC20._transfer(address,address,uint256)
- ButtonswapERC20.balanceOf
- ButtonswapPair.sendOrRefundFee()
- _mint(to,liquidityOut)
- totalSupply = totalSupply + value ButtonswapERC20.totalSupply can be used in cross function reentrancies:
- ButtonswapERC20._burn(address,uint256)
- ButtonswapERC20._mint(address,uint256)
- ButtonswapPair._mintFee(uint256,uint256,uint256,uint256)
- ButtonswapPair.burn(uint256,address)
- ButtonswapPair.burnFromReservoir(uint256,address)
- ButtonswapPair.mint(uint256,uint256,address)
- ButtonswapPair.mintWithReservoir(uint256,address)
- ButtonswapERC20.totalSupply

Reentrancy in ButtonswapPair.mint(uint256,uint256,address): External calls:

- SafeERC20.safeTransferFrom(IERC20(token0),msg.sender,address(this),amountIn0)
- SafeERC20.safeTransferFrom(IERC20(token1),msg.sender,address(this),amountIn1) State variables written after the call(s):
- _mint(address(0),MINIMUM_LIQUIDITY)
- balanceOf[to] = balanceOf[to] + value ButtonswapERC20.balanceOf can be used in cross function reentrancies:
- ButtonswapERC20._burn(address,uint256)
- ButtonswapERC20._mint(address,uint256)
- ButtonswapERC20._transfer(address,address,uint256)
- ButtonswapERC20.balanceOf
- ButtonswapPair.sendOrRefundFee()
- _mint(to,liquidityOut)
- balanceOf[to] = balanceOf[to] + value ButtonswapERC20.balanceOf can be used in cross function reentrancies:
- ButtonswapERC20._burn(address,uint256)
- ButtonswapERC20._mint(address,uint256)
- ButtonswapERC20._transfer(address,address,uint256)
- ButtonswapERC20.balanceOf
- ButtonswapPair.sendOrRefundFee()
- _mint(address(0),MINIMUM_LIQUIDITY)
- totalSupply = totalSupply + value ButtonswapERC20.totalSupply can be used in cross function reentrancies:
- ButtonswapERC20._burn(address,uint256)
- ButtonswapERC20._mint(address,uint256)
- ButtonswapPair._mintFee(uint256,uint256,uint256,uint256)
- ButtonswapPair.burn(uint256,address)
- ButtonswapPair.burnFromReservoir(uint256,address)
- ButtonswapPair.mint(uint256,uint256,address)
- ButtonswapPair.mintWithReservoir(uint256,address)
- ButtonswapERC20.totalSupply
- _mint(to,liquidityOut)

- totalSupply = totalSupply + value ButtonswapERC20.totalSupply can be used in cross function reentrancies:
- ButtonswapERC20._burn(address,uint256)
- ButtonswapERC20._mint(address,uint256)
- ButtonswapPair._mintFee(uint256,uint256,uint256,uint256)
- ButtonswapPair.burn(uint256,address)
- ButtonswapPair.burnFromReservoir(uint256,address)
- ButtonswapPair.mint(uint256,uint256,address)
- ButtonswapPair.mintWithReservoir(uint256,address)
- ButtonswapERC20.totalSupply

8. Appendix C. Tests

Ran 7 test suites: 128 tests passed, 0 failed, 0 skipped (128 total tests)

File	% Lines	% Statements	% Branches	% Funcs
src/ButtonswapERC20.sol	100.00% (28/28)	100.00% (30/30)	100.00% (6/6)	100.00% (8/8)
src/ButtonswapFactory.sol	100.00% (77/77)	100.00% (91/91)	100.00% (34/34)	100.00% (16/16)
src/ButtonswapPair.sol	96.52% (222/230)	97.07% (265/273)	88.54% (85/96)	100.00% (22/22)
src/libraries/Math.sol	12.50% (6/48)	14.89% (7/47)	12.50% (1/8)	33.33% (1/3)
src/libraries/PairMath.sol	0.00% (0/28)	0.00% (0/32)	100.00% (0/0)	0.00% (0/7)
src/libraries/UQ112x112.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
test/ButtonswapPair-MockUFragments.t.sol	0.00% (0/2)	0.00% (0/4)	100.00% (0/0)	0.00% (0/2)
test/mocks/MockButtonswapERC20.sol	100.00% (2/2)	100.00% (2/2)	100.00% (0/0)	100.00% (2/2)
test/mocks/MockButtonswapFactory.sol	31.03% (18/58)	29.58% (21/71)	5.00% (1/20)	25.00% (4/16)
test/mocks/MockButtonswapPair.sol	100.00% (8/8)	100.00% (9/9)	100.00% (2/2)	100.00% (2/2)
test/utils/PairMathExtended.sol	80.00% (4/5)	85.71% (6/7)	100.00% (0/0)	66.67% (2/3)
test/utils/PriceAssertion.sol	85.37% (35/41)	84.00% (42/50)	54.17% (13/24)	50.00% (2/4)
test/utils/Utils.sol	42.86% (6/14)	50.00% (10/20)	40.00% (4/10)	50.00% (2/4)
Total	74.77% (406/543)	75.71% (483/638)	73.00% (146/200)	67.03% (61/91)
--	------------------	------------------	------------------	-----------------

STATE
MIND