Protocol Audit Report

Version 1.0

Cyfrin.io

January 20, 2024

Protocol Audit Report

Duncan DuMond

January 19, 2024

Prepared by: Duncan DuMond Lead Security Researcher: Duncan DuMond - DuncanDuMond

Table of Contents

- Medium
  * [M-1] `TSwapPool::deposit` is missing deadline check causing trnsactions to complete even after the deadline
- Low
  * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
  * [L-2] Default value returned by `TSwapPool::swapExactInput` is results in incorrect value given
- Informational
  * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  * [I-2] Lacking zero address checks
  * [I-3] `PoolFactory::liquidityTokenSymbol` should use `symbol()` instead of `.name()`
  * [I-4] Event is missing `indexed` fields
  * [I-5] MINIMUM_WETH_LIQUIDITY in `TSwapPool::deposit` function is a constant and therefore not required to be emitted
  * [I-6] `poolTokenReserves` in `TSwapPool::deposit` function is a gas line that is not needed
  * [I-7] `liquidityTokensToMint` in `TSwapPool::deposit` function should be added before `_addLiquidityMintAndTransfer` function call
  * [I-8] The parameters in `_mint` of `TSwapPool::_addLiquidityMintAndTransfer` function are backwards
  * [I-9] The `inputAmountMinusFee` and `denominator` variables in `TSwapPool::getOutputAmountBasedOnInput` contain magic numbers. Magic numbers are also shown in the **`return`** statement of the function.
  * [I-10] There is no natspec documentation for `TSwapPool::swapExactInput` function
- Gas
  * [G-1] `TSwapPool::swapExactInput` function should be external to prevent using extra gas
  * [G-2] `TSwapPool::totalLiquidityTokenSupply` function should be external to prevent using extra gas

Protocol Summary

The TSwap protocol is a decentralized exchange protocol that allows users to swap tokens in a trustless manner. The protocol is based on the Uniswap v2 protocol and is deployed on the Ethereum mainnet.

Disclaimer

The New Era Ventures team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope: ./src/ #– PoolFactory.sol #– TSwapPool.sol
- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum

- Tokens:
  - Any ERC20 token

Roles

- Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made.
- Users: Users who want to swap tokens.

Executive Summary

Issues found

Severtity	Number of issues found
High	4
Medium	1
Low	2
Info	10
Gas	2
Total	19

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getOutputAmountBasedOnInput` causes protocol to take too many tokens from users , resulting in lost fees

Description: The `getOutputAmountBasedOnInput` function calculates the fee incorrectly. The fee is calculated as `inputAmountMinusFee * feeNumerator / denominator`, but the `denominator` is calculated as `feeDenominator - feeNumerator`. This means that the fee is calculated as `inputAmountMinusFee * feeNumerator / (feeDenominator - feeNumerator)`, which is incorrect. It scales the amount by 10_000 instead of 1_000. The correct calculation is `inputAmountMinusFee * feeNumerator / feeDenominator`.

Impact: Protocol takes more fees than expected fom users.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10000) / ((
    outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1000) / ((
    outputReserves - outputAmount) * 997);
    }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactInput` causes users to receive less tokens than expected

Description: The `swapExactInput` function does not have slippage protection. This function is similar to what is done in `TSwapPool::swapExactInput` where the function specifies a `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`. This will ensure that users receive the amount of tokens they expect.

Impact: This means that users can receive less tokens than expected. This is because the function does not have a `maxOutputAmount` parameter. This means that the function will execute even if the output amount is less than expected.

Proof of Concept: 1. The price of 1 WETH right now is 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = whatever 3. The function does not offer a `maxInputAmount` parameter, so the function will execute regardless of the input amount 4. As the transaction is pending in the mempool, the market changes! The price moves HUGE -> 1 WETH is not 10,000 USDC. 10x more expensive! 5. The transaction executes, but the user sent the protocol 10,000 USDC instead of 1000 USDC. The user receives 0.1 WETH instead of 1 WETH.

Recommended Mitigation: We should include a `maxInputAmount` parameter in the `swapExactOutput` function. This will ensure that users receive the amount of tokens they expect. The user only has to spend up to a specific amount, and can predict how much they will spend on the protocol.

```
    function swapExactOutput(
        IERC20 inputToken,
        + uint256 maxInputAmount,
        .
        .
        .
    )
    inputAmount = getInputAmountBasedOnOutput(outputAmount,
        inputReserves, outputReserves);
+   if (inputAmount > maxInputAmount) {
+       revert();
+   }

_swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] TSwapPool::sellPoolTokens does not check if the user has enough pool tokens to sell due to mismatching on input and output tokens

Description: The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they are willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculates the swapped amount.

This is due to the fact that the `swapExactOutput` function is called. whereas the `swapExactInput` function is the one that should be called. This is because the user is specifying the exact amount of pool tokens, not output.

Impact: Users will swap the incorrect amount of pool tokens. , which is a severe disruption to protocol functionality.

Proof of Concept: (Try writing in your own words)

Recommended Mitigation:

Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`).

```
    function sellPoolTokens(
        uint256 poolTokenAmount
+       uint256 minWethToReceive,
        ) external returns (uint256 wethAmount) {
```

```
-         return swapExactOutput(i_poolToken, i_wethToken,
    poolTokenAmount, uint64(block.timestamp));
+         return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken
    , minWethToReceive, uint64(block.timestamp));
      }
```

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline parameter.

[H-4] In `TSwawPool::swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`.

Description: The protocol follows a strict invariant of `x * y = k`. Where: - `x` is the amount of input tokens (pool) - `y` is the amount of output tokens (WETH) - `k` is the constant product of the two balances

This means that whenever the balances of the pool change, the ratio between the two amounts should remain constant, hence the `k`. However, this is broken due to the extra incentive in the `_swap` function. Meaning that over time the protocol funds will be drained.

The following block of code is the culprit:

```
-     swap_count++;
-         if (swap_count >= SWAP_COUNT_MAX) {
-             swap_count = 0;
-             outputToken.safeTransfer(msg.sender, 1
    _000_000_000_000_000_000);
-         }
```

Impact: A user could maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra tokens. Simply put, the protocol's core invariant is broken.

Proof of Concept: 1. A user swaps 10 times, and collects the extra incentive of `1_000_000_000_000_000_000` WETH. 2. That user continues to swap until all the protocol funds are drained.

Proof of Code

Place the following into `TSwapPool.t.sol`

```
    function testInvariantBroken() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        uint256 outputWeth = 1e17;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        vm.stopPrank();

        int256 startingY = int256(poolToken.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended Mitigation: Remove the extra incentive mechanism. If you want to keep this in, we should account for the change in the `x * y = k` protocol invariant. Or, we should set aside tokens in the same way we do with fees.

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing trnsactions to complete even after the deadline

Description: The `deposit` function accepts a deadline parameter, which according to the documentation is "The deadline for the transaction to be completed by". However, this parameter is never used. As a consequence, operations that add liquidity to the pool might be executed at unexpected times, which could lead to unexpected results (in market conditions where the deposit rate in unfavorable).

Impact: Transactions could be sent when market conditions are unfavorable to deposit, even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is not used in the `deposit` function. This means that the transaction will be executed regardless of the deadline.

Recommended Mitigation: Consider making the the following change to the function:

```
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, // LP tokens -> liquidity
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran` , the `wethDeposit` and `poolTokensDeposit` parameters are out of order. The `poolTokensDeposit` value should go in the third parameter position, and the `wethDeposit` value should go in the fourth parameter position.

Impact: Event emission is incorrect, leading to off-chain functions potentially malfunctioning.

Recommended Mitigation:

```
-    emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit
   );
+    emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit
   );
```

[L-2] Default value returned by `TSwapPool::swapExactInput` is results in incorrect value given

Description: The `swapExactInput` function function is expected to return the amount of output tokens received. However, while ie declares the named value `output` it is never assigned a value, nor uses an explicit **`return`** statement.

Impact: This means that the function will return the default value of `output`, which is 0. This is incorrect, as the function should return the amount of output tokens received.

Proof of Concept: (Try writing in your own words) –> test case always results in 0 output

Recommended Mitigation:

```
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-        uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
   inputReserves, outputReserves);
+        output = getOutputAmountBasedOnInput(inputAmount, inputReserves
  , outputReserves);

-         if (outputAmount < minOutputAmount) {
-             revert TSwapPool__OutputTooLow(outputAmount,
 minOutputAmount);
+         if (output < minOutputAmount) {
+             revert TSwapPool__OutputTooLow(output, minOutputAmount);
        }

-         _swap(inputToken, inputAmount, outputToken, outputAmount);
+         _swap(inputToken, inputAmount, outputToken, output);
    }
```

Informational

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed

```
- error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

[I-2] Lacking zero address checks

```
    constructor(address wethToken) {
+        if (wethToken == address(0)) {
+            revert();
+        }
        i_wethToken = wethToken;
    }
```

[I-3] `PoolFactory::liquidityTokenSymbol` should use `symbol()` instead of `.name()`

```
-   string memory liquidityTokenSymbol = string.concat("ts", IERC20(
    tokenAddress).name());
+   string memory liquidityTokenSymbol = string.concat("ts", IERC20(
    tokenAddress).symbol());
```

[I-4] Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

- Found in src/PoolFactory.sol Line: 35

  ```
      event PoolCreated(address tokenAddress, address poolAddress);
  ```

- Found in src/TSwapPool.sol Line: 43

  ```
      event LiquidityAdded(address indexed liquidityProvider,
          uint256 wethDeposited, uint256 poolTokensDeposited);
  ```

- Found in src/TSwapPool.sol Line: 44

  ```
      event LiquidityRemoved(address indexed liquidityProvider,
          uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
  ```

- Found in src/TSwapPool.sol Line: 45

  ```
      event Swap(address indexed swapper, IERC20 tokenIn, uint256
          amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
  ```

[I-5] MINIMUM_WETH_LIQUIDITY in `TSwapPool::deposit` function is a constant and therefore not required to be emitted

Constants are not required to be emitted as events. This is because constants are not variables and therefore cannot change. The event is emitted in the constructor, but this is not necessary.

[I-6] `poolTokenReserves` in `TSwapPool::deposit` function is a gas line that is not needed

The `poolTokenReserves` variable is not used in the `deposit` function. This line should be removed.

[I-7] `liquidityTokensToMint` in `TSwapPool::deposit` function should be added before `_addLiquidityMintAndTransfer` function call

The `liquidityTokensToMint` variable is not called to be used as the `wethToDeposit` parameter in the `_addLiquidityMintAndTransfer` function call. This line should be added before the `_addLiquidityMintAndTransfer` function call.

[I-8] The parameters in `_mint` of `TSwapPool::_addLiquidityMintAndTransfer` function are backwards

The `msg.sender` and `liquidityTokensToMint` parameters are backwards in the `_mint` function call. This line should be changed to the following:

```
-    _mint(msg.sender, liquidityTokensToMint);
+    _mint(liquidityTokensToMint, msg.sender);
```

[I-9] The `inputAmountMinusFee` and `denominator` variables in `TSwapPool::getOutputAmountBasedOnInput` contain magic numbers. Magic numbers are also shown in the `return` statement of the function.

The `inputAmountMinusFee` and `denominator` variables contain magic numbers. These should be replaced with constants.

[I-10] There is no natspec documentation for `TSwapPool::swapExactInput` function

The `TSwapPool::swapExactInput` function is missing natspec documentation. This should be added.

Gas

[G-1] `TSwapPool::swapExactInput` function should be external to prevent using extra gas

The `TSwapPool::swapExactInput` function should be external. This will save gas.

- Found in src/TSwapPool.sol Line: 247

  ```
      function swapExactInput(
  ```

[G-2] `TSwapPool::totalLiquidityTokenSupply` function should be external to prevent using extra gas

The `TSwapPool::totalLiquidityTokenSupply` function should be external. This will save gas.