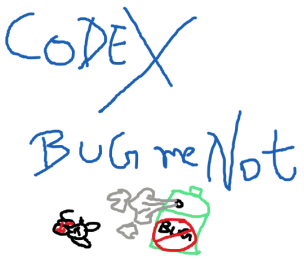

TSwap Protocol Audit Report

Version 1.0

Cyfrin.io

September 6, 2024

Tswap Protocol Audit Report

CodexBugmeNot

September 7th , 2024

Prepared by: [CodexBugmeNot] Lead Auditors: - CodexBugmeNot

Table of Contents

- Mediums
- [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete after the deadline
- Lows
- [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
- [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informationals
  * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  * [I-2] Lacking Zero address checks
  * [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
- [I-4]: Event is missing `indexed` fields

Protocol Summary

This project is meant to be a permissionless way for users to swap assets between each other at a fair price. You can think of T-Swap as a decentralized asset/token exchange (DEX). T-Swap is known as an Automated Market Maker (AMM) because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset. It is similar to Uniswap. To understand Uniswap, please watch this video: Uniswap Explained

Disclaimer

The CodexBugmeNot 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

		Impact		
	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

The Audit went well and the audit was conducted for 11 hours and we used standard tools to conduct the audit.

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	4
Total	11

Findings

Highs

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from usersresulting in too much fee

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of tokens of output tokens. However, the function currently miscalculates the resulting amount. When calculating the fee, it scales the amount by 10_000 instead if 1_000.

Impact: Protocol takes more fees then expected from users

Recommended Mitigation:

```
1
2   function getInputAmountBasedOnOutput(
3           uint256 outputAmount,
4           uint256 inputReserves,
5           uint256 outputReserves
6       )
7           public
8           pure
9           revertIfZero(outputAmount)
10          revertIfZero(outputReserves)
11          returns (uint256 inputAmount)
12      {
13
14  -       return ((inputReserves * outputAmount) * 10000) / ((
        outputReserves - outputAmount) * 997);
15  +       return ((inputReserves * outputAmount) * 10000) / ((
        outputReserves - outputAmount) * 997);
16      }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens

Description: The `swapExactOutput` function deos not include any sort of slippage protection. This function is similar to what is done in `TSwap::swapExactInput` , where the function specifies a `minOutputAmount` , the `swapExactOutput` function should specify a `maxInputAmount`.

Impact: If market conditions change before the transaction processes , the user could get a much worse swap.

Proof of Concept:

1. Price of 1 WETH is 1000 USDC
2. User inputs a `swapExactOutput` amount looking for 1 WETH
3. The function does not offer a max input amount.
4. As the tranaction is in mempool the price of the token changes drastically from 1000 USDC –> 10000 USDC , this is 10 times more than what the user expected
5. The Transaction completes but the user sent 10000 USDC instead of the expected 1000 USDC.

Recommended Mitigation: We should include a `maxInputAmount` so the user only has to spend upto a specific amount, and can predict how much they can spend on the protocol.

```
1
2
3  function swapExactOutput(
4          IERC20 inputToken,
5  +       uint256 maxInputAmount,
6          IERC20 outputToken,
7          uint256 outputAmount,
8          uint64 deadline
9      )
10         public
11         revertIfZero(outputAmount)
12         revertIfDeadlinePassed(deadline)
13         returns (uint256 inputAmount)
14     {
15         uint256 inputReserves = inputToken.balanceOf(address(this));
16         uint256 outputReserves = outputToken.balanceOf(address(this));
17
18         inputAmount = getInputAmountBasedOnOutput(outputAmount,
              inputReserves, outputReserves);
19
20 +        if(inputAmount > maxInputAmount) {
21 +            revert();
```

```
22 +         }
23
24           _swap(inputToken, inputAmount, outputToken, outputAmount);
25       }
```

[H-3] `TswapPool::sellPoolTokens` mismatches input and output tokens causing users to receive incorrect amount of 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 where as the `swapExactInput` function is the one that should be called , because users specify exact amount of input tokens , not output.

Impact: Users will swap the wrong amount of tokens , which is a severs disruption on protocol functionality.

Recommended Mitigation: Consider using `swapExactInput` instead of `swapExactOutput`, Note that this would also require changing the `sellPoolTokens` function to accept a new parameter `minWethToReceive` ti be passed to `swapExactInput` .

```
1
2  -    function sellPoolTokens(uint256 poolTokenAmount ) external returns
       (uint256 wethAmount) {
3  +    function sellPoolTokens(uint256 poolTokenAmount, uint256
       minWethToReceive ) external returns (uint256 wethAmount) {
4
5  }
6  -         return swapExactOutput(i_poolToken, i_wethToken,
       poolTokenAmount, uint64(block.timestamp));
7  +       return swapExactInput(i_poolToken,poolTokenAmount,i_wethToken ,
       minWethToReceive, uint64(block.timestamp));
8       }
```

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

[H-4] In `TSwapPool::_swap` the extra tokens given to the 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 balance of pool token - `y` is the balance of WETH - `k` is the constant product of two balances

This means that whenever the balances change in the protocol, 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 overtime the protocol funds will be completely drained.

The following block of code is responsible for the issue

```
1
2           swap_count++;
3         if (swap_count >= SWAP_COUNT_MAX) {
4             swap_count = 0;
5             outputToken.safeTransfer(msg.sender, 1
                  _000_000_000_000_000_000);
6         }
```

Impact: A user could maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol.

Most simply put the protocols 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` tokens. 2. That user continues untill all the protocol funds are drained

PoC

Place the following into `TSwapPool.t.sol`

```
1
2     function testInvariantBroken() public {
3         vm.startPrank(liquidityProvider);
4         weth.approve(address(pool), 100e18);
5         poolToken.approve(address(pool), 100e18);
6         pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7         vm.stopPrank();
8
9         uint256 outputWeth = 1e17;
10
11        vm.startPrank(user);
12        poolToken.approve(address(pool), type(uint256).max);
13        poolToken.mint(user, 100e18);
14        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
15        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
16        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
17        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
18        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
19        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
```

```
20          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
21          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
22          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
23
24          int256 startingY = int256(weth.balanceOf(address(pool)));
25
26          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
27
28          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
29          vm.stopPrank();
30          uint256 endingY = weth.balanceOf(address(pool));
31          int256 actualDeltaY = int256(endingY) - int256(startingY);
32
33          vm.expectRevert();
34          assertEq(actualDeltaY, expectedDeltaY);
35      }
```

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

```
1
2 -       swap_count++;
3 -         if (swap_count >= SWAP_COUNT_MAX) {
4 -             swap_count = 0;
5 -             outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
6 -         }
```

Mediums

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

Description: The `deposit` function accepts a deadline paramenter, 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 , in market conditions where the deposit rate is unfavourable.

Impact: Transactions could be sent when market conditions are unfavourable to deposit even when adding a deadline paramenter.

Proof of Concept: The `deadline` parameter is unused.

Recommended Mitigation: Consider making the following changes to the function.

```
 function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

Lows

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

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function,it logs values in an incorrect order . The `poolTokensToDeposit` value should go in the third parameter position , whereas the `wethToDeposit` value should go in the second parameter position.

Impact: Event emission is incorrect , leading to offchain functions potentially malfunctioning.

Proof of Concept:

```
 function _addLiquidityMintAndTransfer(
        uint256 wethToDeposit,
        uint256 poolTokensToDeposit,
        uint256 liquidityTokensToMint
    )
        private
    {
        _mint(msg.sender, liquidityTokensToMint);
-     emit LiquidityAdded(msg.sender, poolTokensToDeposit,wethToDeposit)
    ;
+     emit LiquidityAdded(msg.sender, wethToDeposit,poolTokensToDeposit)
    ;
```

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

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

Impact: The return value will always be 0, giving incorrect information to the caller.

Recommended Mitigation:

```
1
2   function swapExactInput(
3         IERC20 inputToken,
4         uint256 inputAmount,
5         IERC20 outputToken,
6         uint256 minOutputAmount,
7         uint64 deadline
8     )
9         public
10        revertIfZero(inputAmount)
11        revertIfDeadlinePassed(deadline)
12 -       returns (uint256 output)
13 +       returns (uint256 outputAmount)
14    {
15        uint256 inputReserves = inputToken.balanceOf(address(this));
16        uint256 outputReserves = outputToken.balanceOf(address(this));
17
18 -        uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount
      , inputReserves, outputReserves);
19 +         outputAmount = getOutputAmountBasedOnInput(inputAmount,
      inputReserves, outputReserves);
20        if (outputAmount < minOutputAmount) {
21            revert TSwapPool__OutputTooLow(outputAmount,
                  minOutputAmount);
22        }
23
24        _swap(inputToken, inputAmount, outputToken, outputAmount);
25    }
```

Informationals

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

```
1
2 - error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

[I-2] Lacking Zero address checks

```
1
2     constructor(address wethToken) {
3
4 +         if(wethToken == address(0)) {
5 +             revert();
6           }
7           i_wethToken = wethToken;
8       }
```

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

```
1 - string memory liquidityTokenSymbol = string.concat("ts", IERC20(
      tokenAddress).name());
2 + 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

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

- Found in src/TSwapPool.sol Line: 43

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

- Found in src/TSwapPool.sol Line: 44

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

- Found in src/TSwapPool.sol Line: 45

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