Protocol Audit Report

Version 1.0

0xJoyBoy03

April 6, 2024

Protocol Audit Report

0xJoyBoy03

Feb 10, 2024

Prepared by: 0xJoyBoy03 Lead Auditors: - *0xJoyBoy03*

Table of Contents

- Medium
  - [M-1] The `TSwapPool::deposit` function is missing deadline check causing the transaction to complete even after the deadline
- Low
  - [L-1] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
  - [L-2] out of order parameters for emit in `TSwapPool::_addLiquidityMintAndTransfer` function
- Gas
  - [G-1] `poolTokenReserves` did not used in `TSwapPool::deposit` function. remove it
- Info
  - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error does not exist and should be remove
  - [I-2] Lacking zero address check in constructor of `PoolFactory` contract
  - [I-3] `PoolFactory::liquidityTokenSymbol` should use `.symbol()` not `.name()`
  - [I-4] 3 events should be indexed in `TSwapPool` contract cause they have more than 3 parameters
  - [I-5] zero address checks in `TSwapPool` constructor
  - [I-6] Constant number should not be emitted in `TSwapPool::deposit` function
  - [I-7] Follow CEI in `TSwapPool::deposit` function at the end
  - [I-9] Magic Numbers in `TSwapPool` contract. create a new stated for `997` and `1000` and `10000`
  - [I-10] There is no natspac for `TSwapPool::SwapExactInput` function
  - [I-11] The `TSwapPool::SwapExactInput` function should be external
  - [\[I-12\] The ‘TSwapPool::SwapExactOutput‘ function is missing ‘deadline‘ @param in its natspac](#i-12-the-tswappoolswapexactoutput-function-is-missing-deadline-param-in-its-natspac)

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 0xJoyBoy03 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
Likelihood	High	H	H/M	M
	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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda ## Scope
- 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

Spended 7 days Auditing this project ## Issues found | Category | No. of Issues | | — | — | | High | 4 | | Medium | 1 | | Low | 2 | | Gas | 1 | | Info | 12 |

Findings

High

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

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 of 1_000.

Impact

Protocol takes more fees than expected from users.

Proof of Concept:

include the following code in the `TSwapPool.t.sol` file:

ZaCode

```
    function test_FlawedSwapByUser() public {
        uint256 initialLiquidity = 100 ether;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit(initialLiquidity, 0, initialLiquidity, uint64(
            block.timestamp));
        vm.stopPrank();

        address alice = makeAddr('alice-chan');
        poolToken.mint(alice, 11 ether);
        vm.startPrank(alice);
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
            timestamp));

        assertLt(poolToken.balanceOf(alice), 1 ether);
        console.log(poolToken.balanceOf(alice));
        vm.stopPrank();

        vm.startPrank(liquidityProvider);
        pool.withdraw(pool.balanceOf(liquidityProvider), 1, 1, uint64(
            block.timestamp));
        vm.stopPrank();

    }
```

Recommend Mitigation

instead of using 10_000, use 1_000 to avoid this

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

Description

In `SwapExactOutput` function there is no `maxInputAmount` parameter to controls the transaction if there are congestion in the blockchain which leads to paying `InputAmount` and receiving fewer `outputAmount` . As we see in `TSwapPool::SwapExactInput` function, there is a `minOutputAmount` parameter for slippage protection, so it should be `maxInputAmount` parameter for `SwapExactOutput` function.

Impact

if the market conditions change before the transaction processes, the user could get a much worse swap!!!

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 maxInput amount
4. As the transaction is pending in the mempool, the market changes! And the price moves HUGE -> 1 WETH is now 10,000 USDC. 10x more than the user expected
5. The transaction completes, but the user sent the protocol 10,000 USDC instead of the expected 1,000 USDC

Recommend Mitigation

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

```
1      function swapExactOutput(
2          IERC20 inputToken,
3          IERC20 outputToken,
4  +       uint256 maxinputAmount,
5
6  .
7  .
8  .
9
10         uint256 inputReserves = inputToken.balanceOf(address(this));
11         uint256 outputReserves = outputToken.balanceOf(address(this));
12 +       if(inputAmount > maxInputAmount) revert();
```

[H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the 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're willing to 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. because users specify the exact amount of input tokens, not

output.

Impact

Users will swap the wrong amount of tokens, which is a severe disruption of protocol functionality.

Proof of Concept:

include the following code in the `TSwapPool.t.sol` file:

ZaCode

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

        vm.startPrank(user);
        uint256 startPoolToken = poolToken.balanceOf(user);
        poolToken.approve(address(pool), type(uint256).max);
        // it fails because of the first high bug we included in our
           report about using `1000` instead of `10000`
        // let's skip it for now
        vm.expectRevert("ERC20: transfer amount exceeds balance");
        pool.sellPoolTokens(10e18);
        assertTrue(startPoolToken - poolToken.balanceOf(user) == 10, "
           User did not sells the specific poolTokens he desired");
        vm.stopPrank();
    }
```

Logs:

```
Ran 1 test for test/unit/TSwapPool.t.sol:TSwapPoolTest
[PASS] testFail_WrongSwapInSellPoolTokens() (gas: 225074)
Logs:
  Error: User did not sells the specific poolTokens he desired
  Error: Assertion Failed

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.72ms
```

Recommend Mitigation

Consider changing the implemetation 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
```

```
4       ) external returns (uint256 wethAmount) {
5   -       return swapExactOutput( i_poolToken, i_wethToken,
        poolTokenAmount, uint64(block.timestamp));
6   +       return swapExactOutput( i_poolToken, poolTokenAmount,
        i_wethToken, minWethToReceive, uint64(block.timestamp));
7       }
```

Note that it might be wise to add a deadline to the function, as there is currently no deadline. MEV stuffs

[H-4] In _swap function the extra tokens given to users after everty `swapcount` resulting in breaks the protocol invariant of `x * y = k`

Description

The protocol follows a strict invariant of `x * y = k`. Where: - `x`: The balance of the pool token - `y`: The balance of WETH - `k`: The constant product of the two balances

This means, that whenever the balances change in the protocol, the ratio betwen the two amounts should remain constant, hence the `k`. However, this is broken due to the extra incentive in the `_swap` function of `TSwapPool` contract. Meaning that over time the protocol funds will be drained. following block of code is responsible for this issue:

```
1 // portion of _swap function :
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

Every 10 swaps the protocol gives the user extra token which leads to breaking of the invariant. 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 protocol's core invariant is broken*.

Proof of Concept:

include the following code in the `TSwapPool.t.sol` file:

ZaCode

```
1       function testFail_CoreInvariantBroked() public {
2           vm.startPrank(liquidityProvider);
3           weth.approve(address(pool), 100e18);
4           poolToken.approve(address(pool), 100e18);
```

```
5           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
6           vm.stopPrank();
7           uint256 X1 = poolToken.balanceOf(address(pool));
8           uint256 Y1 = weth.balanceOf(address(pool));
9           uint256 K1 = X1 * Y1;
10          vm.startPrank(user);
11          poolToken.approve(address(pool), type(uint256).max);
12          for(uint i = 0; i < 10; ++i)
13              pool.swapExactInput(poolToken, 1e17, weth, 0, uint64(block.
                    timestamp));
14          vm.stopPrank();
15          uint256 X2 = poolToken.balanceOf(address(pool));
16          uint256 Y2 = weth.balanceOf(address(pool));
17          uint256 K2 = X2 * Y2;
18          assertTrue(K1 == K2, "Core Invarint broked");
19      }
```

Logs:

```
1  Ran 1 test for test/unit/TSwapPool.t.sol:TSwapPoolTest
2  [PASS] testFail_CoreInvariantBroked() (gas: 413914)
3  Logs:
4    Error: Core Invarint broked
5    Error: Assertion Failed
6
7  Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.44ms
8
9  Ran 1 test suite in 5.44ms: 1 tests passed, 0 failed, 0 skipped (1
       total tests)
```

Recommend Mitigation

Remove the extra incentive.

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

Medium

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

Description

The `deposit` function accepts a deadline parameter, which according to the documetation 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 unfavorable.

Impact

Transactions could be sent when market conditions are unfavorable to deposit, even when adding a dealine parameter,

Proof of Concept: The `deadline` parameter is unused

Recommend Mitigation

Consider making these following changes:

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

Low

[L-1] 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 value`output` but it is never assigned a value, nor uses an explicit return statement.

Impact

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

Proof Of Concepts

Add this following test to the `TSwapPool.t.sol` file:

POC

```
    function test_incorrectInfoInSwapExactInput() public {
        address alice = makeAddr('alice-chan');
        vm.prank(address(pool));
        weth.mint(address(pool), 100 ether);
        vm.startPrank(alice);
        poolToken.mint(alice, 10 ether);
        poolToken.approve(address(pool), 10 ether);
        uint256 output = pool.swapExactInput({
            inputToken: poolToken,
            inputAmount: 10 ether,
            outputToken: weth,
            minOutputAmount: 0,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();
        console.log("Output: ", output);
        console.log("Actual Output: ", weth.balanceOf(alice));
        assertTrue(output != weth.balanceOf(alice));
    }
```

Recommend 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);
    }
```

[L-2] out of order parameters for emit in `TSwapPool::_addLiquidityMintAndTransfer` function

Description

it logs values in an incorrect order

Impact

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

Recommend Mitigation

```
1          _mint(msg.sender, liquidityTokensToMint);
2  -        emit LiquidityAdded(msg.sender, poolTokensToDeposit,
       wethToDeposit);
3  +        emit LiquidityAdded(msg.sender, wethToDeposit,
       poolTokensToDeposit);
```

Gas

[G-1] `poolTokenReserves` did not used in `TSwapPool::deposit` function. remove it

```
1  -    uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```

Info

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error does not exist and should be remove

```
1  contract PoolFactory {
2      error PoolFactory__PoolAlreadyExists(address tokenAddress);
3  -     error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

[I-2] Lacking zero address check in constructor of `PoolFactory` contract

```
1  constructor(address wethToken) {
2  +     require( wethToken != address(0) );
3        i_wethToken = wethToken;
4      }
```

[I-3] `PoolFactory::liquidityTokenSymbol` should use `.symbol()` not `.name()`

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

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

[I-4] 3 events should be indexed in `TSwapPool` contract cause they have more than 3 parameters

```
    event LiquidityAdded(
        address indexed liquidityProvider,
+        uint256 indexed wethDeposited,
+        uint256 indexed poolTokensDeposited
    );
    event LiquidityRemoved(
        address indexed liquidityProvider,
+        uint256 indexed wethWithdrawn,
+        uint256 indexed poolTokensWithdrawn
    );
    event Swap(
        address indexed swapper,
+        IERC20 indexed tokenIn,
+        uint256 indexed amountTokenIn,
+        IERC20 indexed tokenOut,
+        uint256 indexed amountTokenOut
    );
```

[I-5] zero address checks in `TSwapPool` constructor

```
 constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+       require(wethToken != address(0) & poolToken != address(0));
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

[I-6] Constant number should not be emitted in `TSwapPool::deposit` function

```
if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
-                MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
```

[I-7] Follow CEI in `TSwapPool::deposit` function at the end

```
+             liquidityTokensToMint = wethToDeposit;
             _addLiquidityMintAndTransfer(
                 wethToDeposit,
                 maximumPoolTokensToDeposit,
                 wethToDeposit
             );
```

[I-9] Magic Numbers in `TSwapPool` contract. create a new stated for `997` and `1000` and `10000`

```
         // `getOutputAmountBasedOnInput` function. also found in `
            getInputAmountBasedOnOutput`
+         uint256 inputAmountMinusFee = inputAmount * SOME_MAGIC_NUMBER_1;
         uint256 numerator = inputAmountMinusFee * outputReserves;
+         uint256 denominator = (inputReserves * SOME_MAGIC_NUMBER_2) + inputAmountMinusFee;
         return numerator / denominator;
```

[I-10] There is no natspac for `TSwapPool::SwapExactInput` function

```
@>
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {
```

[I-11] The `TSwapPool::SwapExactInput` function should be external

```
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
```

```
5           uint256 minOutputAmount,
6           uint64 deadline
7       )
8   -       public
9   +       external
10          revertIfZero(inputAmount)
11          revertIfDeadlinePassed(deadline)
12          returns (uint256 output)
13      {
```

[I-12] The `TSwapPool::SwapExactOutput` function is missing `deadline` @param in its natspac