Protocol Audit Report

Version 1.0

Aditya Mohan

January 7, 2025

Protocol Audit Report

Aditya Mohan Jha

7 January, 2025

Prepared by: Aditya Lead Security Researcher: - Aditya

Table of Contents

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

- LOW
    * [L-1] `TSwapPool::LiquidityAdded` event has parameter out of order
    * [L-2] Default value return by `TSwapPool::swapExactInput` result in incorrect return value given
- Informational
    * [I-1] `PoolFactory:PoolFactory__PoolDoesNotExist` is not used and should be removed
    * [I-2] `PoolFactory::constructor`Lacking zero address checks
    * [I-3] `PoolFactory::createPool` should call `symbol()` function instead of `name()`
    * [I-4]: Event is missing `indexed` fields
    * [I-5] `TSwapPool::constructor`Lacking zero address checks
    * [I-6] `MINIMUM_WETH_LIQUIDIT` is a constant and not necessary to pass in event `TSwapPool::TSwapPool__WethDepositAmountTooLow`
- GAS
    * [G-1] `TSwapPool::deposit` has the unused variable `poolTokenReserves`

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 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
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

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

Severity	Number of issues found
High	3
Medium	1
Low	2
Info	6
Gas	1
Total	13

Findings

High

[H-1] Incorrect fee calculation in `TSwap::getInputAmountBasedOnOutput` causes protocol to take too many token 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 is currently miss calculating 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.

Recommended Mitigation:

```
function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
```

```
11      {
12  -    return ((inputReserves * outputAmount) * 10000) / ((outputReserves
        - outputAmount) * 997);
13  +    return ((inputReserves * outputAmount) * 1000) / ((outputReserves
       - outputAmount) * 997);
14      }
```

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

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

Impact: If a market condition changes before the transaction process, the user would get a much worse swap.

Proof of Concept: 1. The price of WETH right now is 1000 USDC 2. User input 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 pending in mempool, the market changes! And price move to 1 WETH is now 10000 USDC. 10x more than the user expected. 5. The transaction completed,but user sent the protocol 10000 USDC instead of expected 1000 USDC.

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

```
1
2   function swapExactOutput(
3           IERC20 inputToken,
4   +       uint256 maxInputAmount
5   .
6   .
7   .
8       )
9           inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
10  +        if(inputAmount > maxInputAmount) {
11  +          revert();
12  +        }
13          _swap(inputToken, inputAmount, outputToken, outputAmount);
14      }
```

[H-3] In `TSwapPool::_swap` the extra token given to user after every `SWAP_COUNT_MAX` which breaks the protocol invariant `X*Y=K`

Description: Protocol follows a strict invariant of `x*y=k`. Where:

- `x`: The balance of pool token
- `y`: The balance of WETH token
- `k`: The constant product of two balance

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

Following block of code is responsible for the issue.

```
1     swap_count++;
2
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 funds by doing lot of swap and collecting the extra incentive given out by the protocol.

Most simply put, protocol core invariant is broken.

Proof of Concept: 1. A user swaps 10 times,and collect the extra incentive of `1_000_000_000_000_000_000` tokens. 2. That user continue to swap until all the protocol funds are drains.

POC

```
1   function testInvariantBreak() 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 outputWeth = 1e17;
8           int256 startingY = int256(weth.balanceOf(address(pool)));
9
10          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
11          vm.startPrank(user);
12          poolToken.mint(user,100e18);
13          poolToken.approve(address(pool), type(uint64).max);
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          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
24          vm.stopPrank();
25
26          uint256 endingY = weth.balanceOf(address(pool));
27          int256 actualDeltaY = int256(endingY) - int256(startingY);
28          assertEq(actualDeltaY, expectedDeltaY);
29      }
```

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.

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

Medium

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

Description: The `deposit` function accepts a deadline parameter,which according to the documentation `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 are unfavorable.

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

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

Recommended Mitigation: Consider the following change to the function.

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

LOW

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

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

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

Recommended Mitigation:

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

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

Description: The `swapExactInput` function is expected to return the actual amount of token bought by the caller. However, while it declare the name return value `output` it is never assigned a value not uses an explict return statement.

Impact: Return value always return 0, giving incorrect information to the caller.

Recommended Mitigation:

```
    function swapExactInput(
         IERC20 inputToken,
         uint256 inputAmount,
         IERC20 outputToken,
         uint256 minOutputAmount,
         uint64 deadline
     )
         //written: this should be external
         public
         revertIfZero(inputAmount)
         revertIfDeadlinePassed(deadline)
         //@audit-low:
         // IMPACT: Low - protocol is giving the wrong return
         // LIKELIHOOD: HIGH - always the case
-         returns (uint256 output)
+         returns (uint256 outputAmount)
     {
```

Informational

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

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

[I-2] `PoolFactory::constructor`Lacking zero address checks

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

[I-3] `PoolFactory::createPool` should call `symbol()` function 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.

4 Found Instances

- 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);
```

[I-5] `TSwapPool::constructor`Lacking zero address checks

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

[I-6] `MINIMUM_WETH_LIQUIDIT` is a constant and not necessary to pass in event `TSwapPool::TSwapPool__WethDepositAmountTooLow`

```
- error TSwapPool__WethDepositAmountTooLow(uint256 minimumWethDeposit,
    uint256 wethToDeposit);
+ error TSwapPool__WethDepositAmountTooLow(uint256 wethToDeposit);
```

GAS

[G-1] `TSwapPool::deposit` has the unused variable `poolTokenReserves`

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