TSwap Protocol Audit Report

Version 1.0

nstapleton.com

September 12, 2024

TSwap Protocol Audit Report

nstapleton.com

September 11, 2024

Prepared by: nstap Lead Auditors:

- Niall S

Table of Contents

    * [H-4] In `TSwapPool::_swap_` function tex extra tokens given to users after very `swapCount` breaks the protocol invariant of `x*y=k`
  - Medium
    * [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline
  - Low
    * [L-1] The parameter order is incorrect in the call to emit event `TSwapPool::LiquidityAdded` in `TSwapPool::_addLiquidityMintAndTransfer`
  - Informational
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
    * [I-2] Lacking zero address checks
    * [I-3] `PoolFactory::createPool` should use `symbo()` instead of `name()`
    * [I-4] Event is missing `indexed` fields
    * [I-5] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a constant It does not need to be included the error `TSwapPool::TSwapPool__WethDepositAmountTooLow`

    * [G-1] `poolTokenReserves` is calculated in `TSwapPool::deposit` function unnecessarily

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The nstap 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

```
./src/
#-- PoolFactory.sol
#-- TSwapPool.sol
```

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

Very interesting lesson audit: learnt about re-entrancy, 'weird' ERC tokens, slippage protection and much more.

Issues found

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

Findings

High

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

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

Impact: Protocol takes more fees from users than expected.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
-            ((inputReserves * outputAmount) * 10000) /
+           ((inputReserves * outputAmount) * 1_000) /
            ((outputReserves - outputAmount) * 997);
    }
```

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

Description: The `TSwapPool::swapExactOutput` function does not include any sort of slippage protection. This function is similar to what is done in `TSwapPool::swapExactOutput` 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. The price of WETH is 1,000 USDC
2. user inputs a `swapExactOutput` looking for 1 WETH
    1. inputToken = USDC
    2. outputToken = WETH
    3. outputAmount = 1
    4. deadline = n/a
3. The function does not offer a maxInput amount
4. As the transaction is pending in the memepool, the market changes! And the price moves hugely: 1 WETH is now 10,000 USDC i.e. 10x more than the user expected.
5. The transaction completes but the user send the protocol 10,000 USDC instead of the expecte 1,000 USDC

Recommended 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.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )
        public
        revertIfZero(outputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 inputAmount)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );

+           if(inputAmount > maxInputAmount) {
+               revert();
+           }

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

[H-3] The `TSwapPool::sellPoolTokens` function mismatches input and output tokens causing the user to receive the incorrect amounts of tokens

Description: The `TSwapPool::sellPoolTokens` function is intended to to allow users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to seel 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 tokens.

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

Proof of Concept:

Recommended Mitigation:

consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `swapExactOutput` function to accept a new parameter i.e. `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, there is currently no deadline.

[H-4] In `TSwapPool::_swap_` function tex extra tokens given to users after very `swapCount` breaks the protocol invariant of `x*y=k`

Description: The protocol follows a strict invariant of `x*y=k` Where:

- `x`: balance of the pool token
- `y`: balance of WETH
- `k`: The constant product of the 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 over time the protocal funds will be drained.

The following block of code is responsible for the issue:

```
        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 incentive given out by the protocol.

Proof of Concept:

1. A user swaps 10 times, and collect the extra incentive of `1_000_000_000_000_000_000`
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 {
        uint256 outputWeth = 1e17;

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

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

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        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();

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

Recommended Mitigation: Remove the extra incentive. If it has to stay, 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.

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

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing transactions 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, 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 parameter.

Proof of Concept: The `deadline` parameter is unused

Recommended Mitigation: Consider making 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] The parameter order is incorrect in the call to emit event `TSwapPool::LiquidityAdded` in `TSwapPool::_addLiquidityMintAndTransfer`

Description: The emit statement sends `poolTokensToDeposit` before `wethToDeposit`. The event definition is the other way around.

Impact: Low impact but the values emitted by the event will be wrong

Recommended Mitigation:

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

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::createPool` should use `symbo()` 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

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

- Found in src/TSwapPool.sol Line: 52

  ```
      event LiquidityAdded(
  ```

- Found in src/TSwapPool.sol Line: 57

  ```
      event LiquidityRemoved(
  ```

- Found in src/TSwapPool.sol Line: 62

  ```
      event Swap(
  ```

[I-5] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a constant It does not need to be included the error `TSwapPool::TSwapPool__WethDepositAmountTooLow`

Description: `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a constant. It does not need to be included the error `TSwapPool::TSwapPool__WethDepositAmountTooLow`

Impact: NIL

Recommended Mitigation:

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

[G-1] `poolTokenReserves` is calculated in `TSwapPool::deposit` function unnecessarily

Description: `poolTokenReserves` is calculated but never used in the function. This is unneccessary gas consumption

Impact: NIL

Recommended Mitigation:

```
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
                MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
        }
        if (totalLiquidityTokenSupply() > 0) {
            uint256 wethReserves = i_wethToken.balanceOf(address(this))
                ;
-             uint256 poolTokenReserves = i_poolToken.balanceOf(address(
    this));
```