TSwap Protocol Audit Report

Version 1.0

Sunny Thakur

September 22, 2024

Protocol Audit Report

Prepared by: Sunny Thakur Lead Auditors

Table of Contents

* *[I-3] PoolFacotry::createPool should use .symbol() instead of .name() * [I-4] Event is missing indexed fields*

Introduction

The TSwap Protocol Audit presents a thorough examination of the TSwap platform, a decentralized exchange (DEX) designed to facilitate seamless token swaps and liquidity provision in the blockchain ecosystem. As decentralized finance (DeFi) continues to grow rapidly, ensuring the security and efficiency of platforms like TSwap is essential for maintaining user trust and safeguarding assets.

This audit aims to provide a detailed assessment of the TSwap Protocol's smart contracts, underlying architecture, and operational processes. By identifying potential vulnerabilities and inefficiencies, the audit serves as a critical tool for developers, investors, and users alike, enhancing the overall robustness of the protocol.

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.

Executive Summary

Issues found

Severity	*Number of issues found*
High	*5*
Medium	*0*
Low	*2*
Info	*3*
Total	*10*

Findings

High

[H-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, operationrs 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 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, // LP tokens -> if empty, we can pick 100% (100%
                    == 17 tokens)
                uint256 maximumPoolTokensToDeposit,
                uint64 deadline
        )
                external
```

```
+           revertIfDeadlinePassed(deadline)
            revertIfZero(wethToDeposit)
            returns (uint256 liquidityTokensToMint)
        {
```

[H-2] Incorrect fee calculation in TSwapPool::getInputAmountBasedOnOutput causes protocll 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.*

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10_000) / (( outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1_000) / (( outputReserves - outputAmount) * 997);
    }
```

[H-3] Lack of slippage protection in TSwapPool::swapExactOutput causes users to potentially receive way fewer tokens

Description: *The swapExactOutput function does not include any sort of 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.* ***Impact:*** *If market conditions change before the transaciton processes, the user could get a much worse swap.*

ProofofConcept: Current Market Conditions:

The current price of 1 WETH is 1,000 USDC.

User Transaction Details: 2. A user initiates a swap using the swapExactOutput function, aiming to acquire 1 WETH:

Input Token: USDC

Output Token: WETH

Output Amount: 1 WETH

Deadline: Any valid timestamp (flexible)

Key Observations: 3. The protocol does not specify a maxInput amount for this transaction, leaving the user vulnerable to unexpected outcomes. 4. While the transaction is pending in the mempool, a drastic market change occurs:

The price of 1 WETH skyrockets to 10,000 USDC—a 10x increase from the initial price.

Transaction Outcome: 5. Upon completion of the transaction, the user inadvertently sends the protocol 10,000 USDC instead of the anticipated 1,000 USDC due to the rapid price change.

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*.

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

[H-4] 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 sell in the pool Token Amount parameter. However, the function currently miscalculaes the swapped amount.*

This is due to the fact that the swap Exact Output function is called,where as the swap Exact Input 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 protcol functionality.*

Recommended Mitigation:

Consider changing the implementation to use swapExactInput instead of swapExactOutput. Notethatthiswouldalsorequirechangingthesell PoolTokensfunctiontoacceptanewparameter (ie minWethToReceive to be passed to swapExactInput)

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

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline. (MEV later)

*[H-5] In TSwapPool::_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: 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 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 follow block of code is responsible for the issue.

```
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               }
```

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 protocol's core invariant is broken.

ProofofConcept: *1. Auserswaps10times,andcollectstheextraincentiveof1_000_000_000_000_000_000 tokens 2. That user continues to swap untill all the protocol funds are drained*

Proof Of Code

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.balance   Of(address(p      ool)));
25              int256 expectedDeltaY = int256(-1) * int256(outputWeth);
26
27                                      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                    timestamp));
28              vm.stopPrank();
29
30              uint256 endingY = weth.balanceOf(address(pool));
31              int256 actualDeltaY = int256(endingY) - int256(startingY);
32              assertEq(actualDeltaY, expectedDeltaY);
33      }
```

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  -            // Fee-on-transfer
3  -            if (swap_count >= SWAP_COUNT_MAX) {
4  -                    swap_count = 0;
5  -                    outputToken.safeTransfer(msg.sender, 1 _000_000_000_000_000_000);

6  -            }
```

Low

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

Description: *WhentheLiquidityAddedeventisemittedintheTSwapPool::_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 second.*

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 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 ouput it is never assigned a value, nor uses an explict return statement.*

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

***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 (output < minOutputAmount) {
-                 revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);

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

            }

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

Informationals

[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] PoolFacotry::createPool should use .symbol() instead of .name()

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

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