Protocol Audit Report

Version 1.0

Starkxun

June 20, 2025

Protocol Audit Report

Starkxun

june 29, 2025

Prepared by: Starkxun Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Starkxun 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

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	2

Severity	Number of issues found
Low	2
Info	9
Total	17

Findings

Hign

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol 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:

Code

```
function testFlawedSwapExactOutput() public {
    uint256 initialLiquidity = 100e18;
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), initialLiquidity);
    poolToken.approve(address(pool), initialLiquidity);

    pool.deposit({
        wethToDeposit: initialLiquidity,
        minimumLiquidityTokensToMint: 0,
        maximumPoolTokensToDeposit: initialLiquidity,
        deadline: uint64(block.timestamp)
    });
    vm.stopPrank();

    // User has 11 pool tokens
    address someUser = makeAddr("someUser");
    uint256 userInitialPoolTokenBalance = 11e18;
    poolToken.mint(someUser, userInitialPoolTokenBalance);
    vm.startPrank(someUser);
```

```
20
21      // Users buys 1 WETH from the pool, paying with pool tokens
22      poolToken.approve(address(pool), type(uint256).max);
23      pool.swapExactOutput(
24          poolToken,
25          weth,
26          1 ether,
27          uint64(block.timestamp)
28      );
29
30      // Initial liquidity was 1:1, so user should have paid ~1 pool
            token
31      // However, it spent much more than that. The user started with 11
            tokens, and now only has less than 1.
32      assertLt(poolToken.balanceOf(someUser), 1 ether);
33      vm.stopPrank();
34
35      // The liquidity provider can rug all funds from the pool now,
36      // including those deposited by user.
37      vm.startPrank(liquidityProvider);
38      pool.withdraw(
39          pool.balanceOf(liquidityProvider),
40          1, // minWethToWithdraw
41          1, // minPoolTokensToWithdraw
42          uint64(block.timestamp)
43      );
44
45      assertEq(weth.balanceOf(address(pool)), 0);
46      assertEq(poolToken.balanceOf(address(pool)), 0);
47  }
```

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  -       return ((inputReserves * outputAmount) * 10_000) / ((
        outputReserves - outputAmount) * 997);
14  +       return ((inputReserves * outputAmount) * 1_000) / ((
        outputReserves - outputAmount) * 997);
15      }
```

[H-3] Lack of slippage protection in `TSwapPool::swapExactOutput` causes uses 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 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
5. The function does not offer a maxInput amount
6. 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
7. The transaction completes, but the user sent the protocol 10,000 USDC instead of the expected 1,000 USDC

Code

```
    function testSwapExactOutputSlippageAttack() public {
        uint256 initialLiquidity = 100e18;

        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit({
            wethToDeposit: initialLiquidity,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

        address victim = makeAddr("victim");
        address attacker = makeAddr("attacker");

        weth.mint(victim, 100e18);
        poolToken.mint(victim, 2000e18);
        weth.mint(attacker, 100e18);
        poolToken.mint(attacker, 200e18);
```

```
22
23          vm.startPrank(victim);
24          poolToken.approve(address(pool), type(uint256).max);
25          weth.approve(address(pool), type(uint256).max);
26          uint256 outputAmountWeth = 10e18;
27
28          uint256 beforeInput = pool.getInputAmountBasedOnOutput(
29              outputAmountWeth,
30              poolToken.balanceOf(address(pool)),
31              weth.balanceOf(address(pool))
32          );
33          vm.stopPrank();
34
35          vm.startPrank(attacker);
36          poolToken.approve(address(pool), type(uint256).max);
37          pool.swapExactInput(
38              poolToken,
39              50e18,
40              weth,
41              1e18,
42              uint64(block.timestamp)
43          );
44          vm.stopPrank();
45
46          vm.startPrank(victim);
47          uint256 balanceBefore = poolToken.balanceOf(victim);
48
49          uint256 inputPaid = pool.swapExactOutput(
50              poolToken,
51              weth,
52              outputAmountWeth,
53              uint64(block.timestamp)
54          );
55
56          uint256 balanceAfter = poolToken.balanceOf(victim);
57          vm.stopPrank();
58
59          assertEq(
60              inputPaid,
61              balanceBefore - balanceAfter,
62              "Paid amount and deducted balance mismatch"
63          );
64
65          assertGt(
66              inputPaid,
67              beforeInput + 1e16,
68              "Victim did not overpay significantly after attack"
69          );
70
71          uint256 slippageBps = ((inputPaid - beforeInput) * 10_000) /
               beforeInput;
```

```
72          console.log("Slippage (basis points):", slippageBps);
73          assertGt(slippageBps, 500, "Slippage less than 5%, attack
                impact too small");
74      }
```

Recommended Mitigation: We should include a `maxInputAmount` so the user only has to spend up to a specific amoun

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

[H-4] `TSwaPool::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 `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 tokens, not output.

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

Proof of Concept:

Code

```
1       function testSellPoolTokensSwapsCorrectly() public {
2           uint256 initialLiquidity = 100e18;
3
4           // set LP
5           vm.startPrank(liquidityProvider);
6           weth.approve(address(pool), initialLiquidity);
```

```
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit({
            wethToDeposit: initialLiquidity,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

        // user try to sell pool tokens
        vm.startPrank(user);
        uint256 wethBefore = weth.balanceOf(user);
        uint256 poolTokensToSell = 10e18;

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

        uint256 receivedWeth = pool.sellPoolTokens(poolTokensToSell);
        uint256 wethAfter = weth.balanceOf(user);
        uint256 actualReceived = wethAfter - wethBefore;

        assertEq(receivedWeth, actualReceived, "Returned WETH does not
            match actual received");
        vm.stopPrank();
    }
```

Recommended Mitigation: Consider changing the implementation 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,
        ) 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, as there is currently no deadline. (MEV later)

[H-5] ### `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.

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 collects the extra incentive of 1_000_000_000_000_000_000 tokens
2. That user continues to swap until all the protocol funds are drained

Code

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

        uint256 outputWeth = 1e17;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);
        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));

```

```
        int256 startingX = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaX = int256(outputWeth) * -1;

        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        vm.stopPrank();

        int256 endingX = int256(weth.balanceOf(address(pool)));
        int256 actualDeltaX = int256(endingX) - int256(startingX);

        assertEq(actualDeltaX, expectedDeltaX);
    }
```

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.

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

[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 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 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,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
```

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

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emit incorrect information.

Description: What 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 second.

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

Proof of Concept:

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 `output` 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.

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] `PoolFactory::constructor` Lacking zero address check

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

[I-3] `PoolFactory::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());
```