TSwap Protocol Audit Report

Version 1.0

AlbertoGuirado.com

July 23, 2024

T-SWAP Audit - Invariants

Tastukesi

July 16, 2024

Prepared by: [Alberto] Lead Auditors: - Cyfrin

Table of Contents

    * [H-4] The function `TSwapPool::sellPoolTokens` mismatches input and output tokens causing a wrong call: users to receive the incorrect amount of tokens
    * [H-5] In `TSwapPool::_swap` the extra tokens fiven to users after every `swapCount` breaks the protocol invariant of `x*y=k`
  - MEDIUM
    * [M-2] Rebase, fee-on-transfer, and ERC777 tokens breaks the protocol invariant
  - LOW
    * [L-1] `TSwapPool::LiquidityAdded` event has parameter out of order
    * [L-2] Default value of " results
  - INFORMATIONAL
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` should be removed
    * [I-2] Lacking zero address check
    * [I-3] Wrong call to a function
    * [I-4] Elements of event should be indexed
    * [I-5] The constant `MINIMUM_WETH_LIQUIDITY` shouldn't be emmited

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The YOUR_NAME_HERE 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

		Impact		
	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

Severtity	Numb of issues found
High	4
Medium	2
Low	2
Info	9
Total	17

Risk Classification

		Impact		
		High	Medium	Low

		Impact		
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

Findings

HIGH

[H-1] `TswapPool::deposit` is missing deadline check causing transaction 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 a liquidity to the pool might be executed at unexpeted itimes, in maket condigionts where the deposit reate 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

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )external
    revertIfZero(wethToDeposit)
+   revertIfDeadlinePassed(deadline)
    return (uint liquidityTokensToMint)
    {...}
```

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens, 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 from users

Proof of concept

```
function testFlawedSwapExactOutput() public {
        uint256 initialLiquidity = 100e10;
        vm.startPrank(liquidityProvider);
            weth.approve(address(pool), initialLiquidity);
            poolToken.approve(address(pool), initialLiquidity);

            pool.deposit({
                wethToDeposit: initialLiquidity,
                minimumLiquidityTokensToMint: 0,
                maximumPoolTokensToDeposit: 2e11,
                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);
            poolToken.approve(address(pool), type(uint).max);
            //Initial liquidity was 1:1, so user should have paid around 1 poolToken
            // However, it spent much more than that. The user started with 11 token and now has only less than 2
            pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.timestamp));
            assertLt(poolToken.balanceOf(someUser),1 ether);
        vm.stopPrank();

        //The liquidity provider can rug all funds from the pool now,
        // including thos deposited by user
        vm.startPrank(liquidityProvider);
        pool.withdraw(
            pool.balanceOf(liquidityProvider),
            1,
            1,
            uint64(block.timestamp));

        assertEq(weth.balanceOf(address(pool)),0);
        assertEq(poolToken.balanceOf(address(pool)), 0);

    }
```

Recommened Mitigation

```diff
-     return ((inputReserves * outputAmount) * 10_000) / ((outputReserves - outputAmount) * 997);

+     return ((inputReserves * outputAmount) * 1_000) / ((outputReserves - outputAmount) * 997);
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput`. Need a max value amount parameter. Causes users to potentially reveive way fewer tokens

Description The `swapExactOutput` function does not include any sort of slippage protection. (Search in SOLODIT) This function is similar to what is done in `TSwapPool::swapWxactInput` where the function specifies a `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmoint`.

- Here is 10 WETH -> Give me DAI
- Here 10 WETH -> At least 100 DAI
- How much WETH do I need to give give you, *ill do a max of 10* WETH-> 100 DAI ACTUAL SITUATION
- Here is 10W -> Gimme the DAI equivalent
- I want 10 DAI, charge me as much WETH as need

Impact If market conditions change before the transaction processess, the user could get a much worse swap. An attacker could do a fornt attack or sandwich attack that could change the price before the purchase

Proof of concept 1. The price of WETH right now is 1_000 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. output = 1 4. deadline whatever 3. The function does not offer a maxInput amojunt 4. As the transaction is pending in the memPool, the market changes! And the price moves HUGE -> 1 WETH is now 10_000 USDC. 5. The transaction completes, but the user sent the protocol 10_000 USDC instead of the expected 1_000USDC.

Recommened 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.

```diff
    funtion(
+       uint256 maxInputAmount

    ){
.
.
```

```
    inputAmount = getInputAmountBasedOnOutput(outputAmount,
       inputReserves, outputReserves);
+       if(inputAmount > maxInputAmount) revert();
        _swap(inputToken, inputAmount, outputToken, outputAmount);
    }
```

[H-4] The function `TSwapPool::sellPoolTokens` mismatches input and output tokens causing a wrong call: 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 input tokens, not output.

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

Proof of concept

Recommened Mitigation Consider changing the implementation: Not that this

```
function sellPoolTokens(
        uint256 poolTokenAmount,
+       uint256 minWethToReceive,
    ) external returns (uint256 wethAmount) {
      +        return
-            swapExactOutput(
-                i_poolToken, // pt
-                i_wethToken, // wt
-                poolTokenAmount,
-               uint64(block.timestamp)
-            );
+        return
+            swapExactOutput(
+                i_poolToken, // pt
+                i_wethToken, // wt
+                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] In `TSwapPool::_swap` the extra tokens fiven 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`: Balance of WETH - `k`: The constant product of the two balances

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

The following block of code is responsible of 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 maliciousy drain the protocol of funds by doing a lot of swap and collecting the extra incentive given out by the protocol. Most simply but, the protocol's core invariant is broken.

Proof of Concept 1. A user swap 10 times, and collects the extra incentive of `1_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`

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

        vm.startPrank(user);
        // Approve tokens so they can be pulled by the pool during the swap
        poolToken.approve(address(pool), type(uint256).max);
        //poolToken.mint(address(pool), amount);
        // Execute swap, giving pool tokens, receiving WETH
        pool.swapExactOutput({
            inputToken: poolToken,
            outputToken: weth,
            outputAmount: outputWeth,
            deadline: uint64(block.timestamp)
        });

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

        vm.stopPrank();

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

Recommended Mitigations Remove the extra incentive. 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++;
-        if (swap_count >= SWAP_COUNT_MAX) {
-            swap_count = 0;
-            outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
-        }
```

MEDIUM

[M-2] Rebase, fee-on-transfer, and ERC777 tokens breaks the protocol invariant

///findings…

LOW

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

Description When the `LoquidityAdded` evetn is emmited in the funciont, it logs values in an incorrrect order. The `poolTokenToDeposit` value should go in the third parameter position

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 of “ results

Description The `swapExtactInput` function is expected to return the actual amount of tokens bought by the caller. However, while it declares the named return value `output` its never assined a value, nor uses an explict return statement.

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

Proof of concept <>

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 (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);
     }
```

INFORMATIONAL

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` should be removed

Description This function is not used and should be removed.

Impact

Proof of concept

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

[I-2] Lacking zero address check

Proof of concept

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

[I-3] Wrong call to a function

Description Wrong call to function. Should be `.symbol` not `.name` **Proof of concept**

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

[I-4] Elements of event should be indexed

Description The elements of events should be indexed when there're more than 3 **Proof of concept**

```
event Swap(
        address indexed swapper,
        IERC20 tokenIn,
        uint256 amountTokenIn,
        IERC20 tokenOut,
        uint256 amountTokenOut
    );
```

[I-5] The constant `MINIMUM_WETH_LIQUIDITY` shouldn't be emmited

Could cause waste of gas

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