T Swap Audit Report

Version 1.0

Borken Angel

November 23, 2024

T Swap Audit Report

BrokenAngel

Nov 23, 2024

Prepared by: BrokenAngel

Lead Security Researcher:

- BrokenAngel

Table of Contents

    * [H-5] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`

- Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
    * [L-3] **`public`** functions not used internally could be marked `external`
    * [L-4] Define and use `constant` variables instead of using literals
    * [L-5] Event is missing `indexed` fields
    * [L-6] PUSH0 is not supported by all chains
    * [L-7]: Large literal values multiples of 10000 can be replaced with scientific notation
    * [L-8]: Unused Custom Error
- Informationals
    * [I-1] Lacking zero address checks
    * [I-2] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`

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 BrokenAngel 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

The findings decribed in this document correspond the foloowing commit hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

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

Issues found

Severity	Number of issue found
High	5
Medium	0
Low	8
Info	2
Total	15

Findings

High

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

Description: The `deposit` functions 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 unexcepted 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 paramter. **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 ->
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadline(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[H-2] Protocol may take too many tokens from users during swap, resulting is lost fee

Description: The `getInputAmountBasedOnOutput` function is intented to calculate the amount of tokens a user should deposit given an amount of output tokens. However, the function currently miscalculate the resulting amount. When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: As a result, users swapping tokens via the `swapExactOutput` function will pay far more tokens than expected for their trades. This becomes particularly risky for users that provide infinite allowance to the `TSwapPool` contract. Morever, note that the issue is worsened by the fact that the `swapExactOutput` functions does not allow users to specify a maximum of input tokens, as is described in another issue in this report.

It's worth noting that the tokens paid by users are not lost, but rather can be swiftly taken by liquidity providers. Therefore, this contract could be used to trick users, have them swap their funds at unfavorable rates and finally rug pull all liquidity from the pool.

Proof of Concept:

To test this, include the following code in the `TSwapPool.t.sol` file:

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

        pool.deposit(
            initialLiquidity,
            0,
            initialLiquidity,
            uint64(block.timestamp)
        );
        vm.stopPrank();

        address someUser = makeAddr("SomeUser");
        uint256 userInitialPoolTokenBalance = 11e18;
        poolToken.mint(someUser, userInitialPoolTokenBalance);

        // User buys 1 WETH from the pool, paying with pool tokens
        vm.startPrank(someUser);
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
            timestamp));

        // Initial liquidity was 1:1, so user should have paid ~1 pool
            token
        // However, it spent much more than that. The user started with
             11 tokens, and now only has less than
        assertLt(poolToken.balanceOf(someUser), 1 ether);
        vm.stopPrank();

        // The liquidity provider can rug all funds from the pool now,
        // including those 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);
```

```
41      }
```

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          swapping functions!!
14          return
15  -               ((inputReserves * outputAmount) * 10000) / ((
        outputReserves - outputAmount) * 997);
16  +               ((inputReserves * outputAmount) * 1000) / ((
        outputReserves - outputAmount) * 997);
17      }
```

[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 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 input a `swapExactOutput` looking for 1 WETH
    i. inputToken = USDC
    ii. outputToken = WETH
    iii. outputAmount = 1
    iv. deadline = whatever
3. The function does not offer a maxInput amount

4. As the transaction is pending the mempool, the market changes! And the price moves HUGE -> 1 WETH is now 10,000 USDC. 10x more than the uer expected
5. The transaction completes but the user sent the protocol 10,000 instead of the expected 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,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(outputAmount,
           inputReserves, outputReserves);
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
        _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 easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell in the `poolToken` parameter. However, the function currently miscalculate 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: User will swap the wrong amount of tokens, which is a severe disruption of protocol functionality.

Proof of Concept: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new paremeter (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] 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 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.

Most simply put, the protocol's core invariant is broken.

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

Proof of Code

Place the following into `TSwapPool.t.sol`.

```
    function testBreakInvariant() public {
        testDeposit();
```

```
3
4           vm.startPrank(user);
5           poolToken.approve(address(pool), type(uint256).max);
6           for (uint256 i = 0; i < 9; i++) {
7               uint256 balanceBefore = weth.balanceOf(user);
8               uint256 outputAmount = pool.getOutputAmountBasedOnInput(
9                   1e18,
10                  poolToken.balanceOf(address(pool)),
11                  weth.balanceOf(address(pool))
12              );
13
14              pool.swapExactInput(
15                  poolToken,
16                  1e18,
17                  weth,
18                  0,
19                  uint64(block.timestamp)
20              );
21              uint256 balanceAfter = weth.balanceOf(user);
22              assertEq(balanceAfter - balanceBefore, outputAmount);
23          }
24
25          uint256 balanceBefore = weth.balanceOf(user);
26          uint256 outputAmount = pool.getOutputAmountBasedOnInput(
27              1e18,
28              poolToken.balanceOf(address(pool)),
29              weth.balanceOf(address(pool))
30          );
31
32          pool.swapExactInput(poolToken, 1e18, weth, 0, uint64(block.
                timestamp));
33          uint256 balanceAfter = weth.balanceOf(user);
34          assertEq(balanceAfter - balanceBefore, outputAmount + 1e18);
35
36          vm.stopPrank();
37      }
```

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

Low

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

Description: When the `LiqudityAdded` event is emitted in the `TSwapPool::_addLiqudityMintAndTransf` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, whereas the `wethToDeposit` valud should go second.

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

Recommended Mitigation:

```
1 -    emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit)
      ;
2 +    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 users an explict return statement.

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

Recommended Mitigation:

```
1
2           uint256 inputReserves = inputToken.balanceOf(address(this));
3           uint256 outputReserves = outputToken.balanceOf(address(this));
4
5 -         uint256 outputAmount = getOutputAmountBasedOnInput(
      inputAmount, inputReserves, outputReserves );
6 +         output = getOutputAmountBasedOnInput(            inputAmount,
      inputReserves, outputReserves );
7 -         if (outputAmount < minOutputAmount) { revert
      TSwapPool__OutputTooLow(outputAmount, minOutputAmount); }
8 +         if (output < minOutputAmount) { revert TSwapPool__OutputTooLow(
      output, minOutputAmount); }
9
10 -        _swap(inputToken, inputAmount, outputToken, outputAmount);
11 +        _swap(inputToken, inputAmount, outputToken, output);
```

[L-3] `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

1 Found Instances

- Found in src/TSwapPool.sol Line: 299

```
    function swapExactInput(
```

[L-4] Define and use `constant` variables instead of using literals

If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

4 Found Instances

- Found in src/TSwapPool.sol Line: 277

```
        uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 296

```
            ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 457

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 466

```
                1e18,
```

[L-5] 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: 52

```
1      event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1      event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1      event Swap(
```

[L-6] PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

2 Found Instances

- Found in src/PoolFactory.sol Line: 15

```
1  pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
1  pragma solidity 0.8.20;
```

[L-7]: Large literal values multiples of 10000 can be replaced with scientific notation

Use `e` notation, for example: `1e18`, instead of its full numeric value.

3 Found Instances

- Found in src/TSwapPool.sol Line: 45

```
1      uint256 private constant MINIMUM_WETH_LIQUIDITY = 1
          _000_000_000;
```

- Found in src/TSwapPool.sol Line: 295

```
1              ((inputReserves * outputAmount) * 10000) /
```

- Found in src/TSwapPool.sol Line: 405

```
1             outputToken.safeTransfer(msg.sender, 1
                  _000_000_000_000_000_000);
```

[L-8]: Unused Custom Error

it is recommended that the definition be removed when custom error is unused

1 Found Instances

- Found in src/PoolFactory.sol Line: 22

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

Informationals

[I-1] Lacking zero address checks

```
1     constructor(address wethToken) {
2         + if(wethToken == address(0)) revert();
3         i_wethToken = wethToken;
4     }
```

[I-2] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`

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