T-Swap Protocol Audit Report

Version 1.0

Lulox

May 30, 2024

T-Swap Protocol Audit Report

Lulox

May 30, 2024

Prepared by: Lulox Lead Auditors:

- Lulox

Table of Contents

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 Lulox 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
Likelihood	High	H	H/M	M
	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/
#-- TSwapPool.sol
#-- PoolFactory.sol
```

Roles

- Liquidity Provider: A user who provides liquidity to the protocol.
- User: A user who interacts with the protocol to swap tokens.

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	8
Gas	2
Total	18

Findings

High

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

```
1   function testFlawedSwapExactOutput() public {
2     uint256 initialLiquidity = 100e18;
3     vm.startPrank(liquidityProvider);
```

```
4       weth.approve(address(pool), initialLiquidity);
5       poolToken.approve(address(pool), initialLiquidity);
6
7       pool.deposit({
8           wethToDeposit: initialLiquidity,
9           minimumLiquidityTokensToMint: 0,
10          maximumPoolTokensToDeposit: initialLiquidity,
11          deadline: uint64(block.timestamp)
12      });
13      vm.stopPrank();
14
15      // User has 11 pool tokens
16      address someUser = makeAddr("someUser");
17      uint256 userInitialPoolTokenBalance = 11e18;
18      poolToken.mint(someUser, userInitialPoolTokenBalance);
19      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      assertEq(weth.balanceOf(address(pool)), 0);
45      assertEq(poolToken.balanceOf(address(pool)), 0);
46  }
```

Recommended Mitigation:

```
1     function getInputAmountBasedOnOutput(
2           uint256 outputAmount,
3           uint256 inputReserves,
```

```
4           uint256 outputReserves
5       )
6           public
7           pure
8           revertIfZero(outputAmount)
9           revertIfZero(outputReserves)
10          returns (uint256 inputAmount)
11      {
12
13  -       return ((inputReserves * outputAmount) * 10_000) / ((
        outputReserves - outputAmount) * 997);
14  +       return ((inputReserves * outputAmount) * 1_000) / ((
        outputReserves - outputAmount) * 997);
15      }
```

[H-2] 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 inputs a `swapExactOutput` transaction to buy 1 WETH
   - inputToken: USDC
   - outputToken: WETH
   - outputAmount: 1 WETH
   - deadline: whatever
3. The function does not offer a `maxInputAmount`.
4. 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.
5. The transaction completes, but the user sent the protocol 10,000 USDC instead of the expected 1,000 USDC

Proof of Code to be written.

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-3] `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 `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 amount of tokens, which is a severe disruption of protocol functionality.

Proof of Concept: To be written

Recommended Mitigation: Consider changing the implementation to use the `swapExactInput` function. Note that this would also require to change the `sellPoolTokens` function to accept a new parameter (e.g., `minWethToReceive`) to be passed down 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 would be wise to add a deadline to the function, as there is currently no deadline. (MEV later)

[H-4] 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++;
// Fee on transfer
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. The user continues to swap until 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;

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

```
12          poolToken.mint(user, 100e18);
13          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
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
23          int256 startingY = int256(weth.balanceOf(address(pool)));
24          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25
26          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31          assertEq(actualDeltaY, expectedDeltaY);
32      }
```

Recommended Mitigation: 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.

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, while 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 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)
     {
```

[M-2] Rebase, fee on transfer and ERC-777 break the protocol invariant of `x * y = k`

Description: Tokens with certain mechanisms break the protocol invariant by their design.

Impact: The protocol invariant is broken, and the protocol could be drained of funds.

Proof of Concept: To be written

Recommended Mitigation: Consider reading Uniswap V1 audit by Consensys Dilligence for more information on how to handle these tokens.

Low

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

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

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

```
function testSwapExactInputReturnValueIsAlwaysZero() 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);

        // Users buys 1 WETH from the pool, paying with pool tokens
        poolToken.approve(address(pool), type(uint256).max);
        uint256 output = pool.swapExactInput(poolToken, 1 ether, weth,
            0, uint64(block.timestamp));

        assertEq(output, 0);
        vm.stopPrank();
    }
```

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);
      }
```

Informationals

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed

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

[I-2] Lacking zero address checks

On `PoolFactory.sol`

```
constructor(address wethToken) {

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

On `TSwapPool.sol`

```
 constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    )
        ERC20(liquidityTokenName, liquidityTokenSymbol)
    {
+       if(wethToken == address(0) || poolToken == address(0)) {
+           revert();
+       }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

[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());
```

[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.

- Found in src/PoolFactory.sol Line: 3+

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

- Found in src/TSwapPool.sol Line: 43

```
    event LiquidityAdded(address indexed liquidityProvider,
        uint256 wethDeposited, uint256 poolTokensDeposited);
```

- Found in src/TSwapPool.sol Line: 44

```
    event LiquidityRemoved(address indexed liquidityProvider,
        uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
```

- Found in src/TSwapPool.sol Line: 45

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

[I-5] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a constant and therefore not required to be emitted in `TSwapPool__WethDepositAmountTooLow` event

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

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

```
+       revert TSwapPool__WethDepositAmountTooLow(wethToDeposit);
    }
```

[I-6] State change should happen before interaction in `TSwapPool::deposit` to follow CEI

```
} else {
+             liquidityTokensToMint = wethToDeposit;
             // This will be the "initial" funding of the protocol. We are starting from blank here!
             // We just have them send the tokens in, and we mint liquidity tokens based on the weth
             _addLiquidityMintAndTransfer(wethToDeposit, maximumPoolTokensToDeposit, wethToDeposit);
-             liquidityTokensToMint = wethToDeposit;
         }
```

[I-7] `TSwapPool::swapExactInput` should have natspec

```
+ /**
+    * @param inputToken Token to swap / sell (ie: DAI)
+    * @param inputAmount Amount of input token to swap / sell
+    * @param outputToken Token to buy / receive (ie: WETH)
+    * @param minOutputAmount Minimum output amount expected to receive
+    * @param deadline When the transaction should expire
+    */
   function swapExactInput(
```

[I-8] 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.

- Found in src/TSwapPool.sol Line: 231

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

- Found in src/TSwapPool.sol Line: 259

```
        return ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 387

```
1                1e18, i_wethToken.balanceOf(address(this)),
                    i_poolToken.balanceOf(address(this))
```

- Found in src/TSwapPool.sol Line: 393

```
1                1e18, i_poolToken.balanceOf(address(this)),
                    i_wethToken.balanceOf(address(this))
```

Gas

[G-1] `poolTokenReserves` variable within `TSwapPool::deposit` is stored in memory and never used

```
1 - uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```

[G-2] `public` functions not used internally could be marked `external`

Description: As the function `TSwapPool::swapExactInput`, `TSwapPool::swapExactOutput` and `TSwapPool::totalLiquidityTokenSupply` are not used internally, they should be marked as external to save up gas when deploying.