TSwap Audit Report

Version 1.0

FalseGenius

May 14, 2024

TSwap Audit Report

FalseGenius

May 14, 2024

Prepared by: FalseGenius Lead Auditors: - None

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.

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

The findings described in this document correspond to the following commit hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

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

Roles

- Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made.
- Users: Users who want to swap tokens.

Executive Summary

I delved into auditing this codebase to learn more about DeFI. I spent 3~4 days using Foundry tests, Slither, Aderyn and Manual Review for the Audit

Issues found

Severity	Number of Issues Found
High	4
Medium	1
Low	3
Informational	4
Total	12

Findings

High

[H-01] Incorrect calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in loss of fees.

Description: The `getInputAmountBasedOnOutput` function is intended to calculate amount of tokens user needs to deposit, given the output tokens the user wants. As per documentation, it should apply a 997 out of 1000 multiplier fee that stays in the protocol. However, it miscalculates the fee by scaling amount by 10000 instead of 1000, charging way too much from the user.

Impact: Protocol takes more fees than expected from the users.

Proof of Concept: Consider the following scenario, 1. User needs 10 WETH. 2. User triggers `getInputAmountBasedOnOutput` to get the expected amount. 3. The amount user expects: 11 pool tokens by following documentation calculations. 4. Actual amount returned: 111 pool Tokens! 5. The protocol expects user to provide 111 pool tokens to get 10 WETH!

PoC

```
1
2     function testIncorrectCalculationIngGetInputAmountBasedOnOutput()
          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 inputReserves = poolToken.balanceOf(address(pool));
10          uint256 outputReserves = weth.balanceOf(address(pool));
11          uint256 outputAmount = 10e18;
12          uint256 expectedAmount = ((inputReserves * outputAmount) *
                1000) /((outputReserves - outputAmount) * 997);
13          // user wants 10 WETH. How much pool token he needs to feed?
14          vm.prank(user);
15          uint256 actualAmount = pool.getInputAmountBasedOnOutput(
                outputAmount, inputReserves, outputReserves);
16
17          // Expected: 11 poolTokens
18          // Actual Amount: 111 poolTokens!
19          console.log("Expected amount: %s", expectedAmount);
20          console.log("poolToken needed for 10 WETH before: %s",
                actualAmount);
21
22          assertTrue(expectedAmount != actualAmount);
23      }
```

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) * 10000) / ((outputReserves
        - outputAmount) * 997);
14  +       return ((inputReserves * outputAmount) * 1000) / ((outputReserves
        - outputAmount) * 997);
15      }
```

[H-02] Lack of Slippage protection in `TSwapPool::swapExactOutput` for `inputAmount`, causing users to potentially receive less tokens.

Description: The `swapExactOutput` does not include any slippage protection. This function is similar to `TSwapPool::swapExactInput`, where user specifies `minOutputAmount` that he wants for the input tokens.

Impact: If market conditions change before transaction processes, user could get much worse swap and get charged way more than expected.

Proof of Concept: Consider this scenario, 1. The price of 1 WETH is 1,000 USDC 2. User inputs token in `swapExactOutput`, looking to get 1 WETH. 1. inputToken = USDC, 2. outputToken = WETH, 3. outputAmount = 1 WETH, 4. deadline = whatever 3. The function doesn't allow maxInputAmount. 4. As transaction is pending in mempool and market prices moves to 1 WETH -> 10,000 USDC, which is 10x more than the user expected. 5. Transaction goes through, and user sents 10,000 for 1 WETH.

Recommended Mitigation:

```
1
2       function swapExactOutput(
3           IERC20 inputToken,
4   +       uint256 maxInputAmount
5           IERC20 outputToken,
6           uint256 outputAmount,
7           uint64 deadline
8       )
9           public
10          revertIfZero(outputAmount)
11          revertIfDeadlinePassed(deadline)
12          returns (uint256 inputAmount)
13      {
14          uint256 inputReserves = inputToken.balanceOf(address(this));
15          uint256 outputReserves = outputToken.balanceOf(address(this));
16
17          // @audit-high; no slippage protection: need a max input amount
18
19          inputAmount = getInputAmountBasedOnOutput(
20              outputAmount,
21              inputReserves,
22              outputReserves
23          );
24
25  +       if (inputAmount > maxInputAmount) revert();
26
27          _swap(inputToken, inputAmount, outputToken, outputAmount);
28      }
```

[H-03] Token mismatch in `TSwapPool::sellPoolTokens`, results in users receiving incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users to easily sell their poolTokens, getting WETH in exchange. Users indicate how many poolTokens they're willing to sell in the `poolTokenAmount` parameter. This function calls `TSwapPool::swapExactOutput` instead of calling out `TSwapPool::swapExactInput` since user specified exact input tokens.

Impact: Users would receive wrong amount of tokens which is a severe disruption of the protocol.

Proof of Concept: 1. A user with balance of 200 pool tokens wants to sell 10 pool tokens. 2. User sells 10 pool tokens using `sellPoolTokens` function and receives more than expected WETH. 3. In addition, the protocol takes 112 pool tokens instead of 10 pool tokens which users is selling!

PoC

```
    function testMismatchInSellPoolTokens() public liquidityAdded {
        address alice = makeAddr("alice");
        poolToken.mint(alice, 200 ether);
        vm.prank(alice);
        poolToken.approve(address(pool), 200 ether);
        uint256 userBalanceBeforeSellingToken = poolToken.balanceOf(
            alice);

        uint256 wethBefore = weth.balanceOf(alice);
        assertEq(wethBefore, 0);

        uint256 inputAmount = 10 ether;
        uint256 expectedAmount = pool.getOutputAmountBasedOnInput(
            inputAmount, poolToken.balanceOf(address(pool)), weth.
            balanceOf(address(pool)));

        vm.prank(alice);
        pool.sellPoolTokens(inputAmount);

        uint256 expectedUserBalanceAfterSellingToken =
            userBalanceBeforeSellingToken - 10 ether;
        uint256 actualUserBalanceAfterSellingToken = poolToken.
            balanceOf(alice);
        uint256 actualAmountReceived = weth.balanceOf(alice);
        // expected:  9.06e18
        // actual:    10e18
        // pool token balance: 88e18 instead of 190e18!
        console.log("ExpectedAmount: %s", expectedAmount);
        console.log("ActualAmount: %s", actualAmountReceived);
        console.log("pool token balance left: %s", poolToken.balanceOf(
            alice));

        assertTrue(expectedAmount != actualAmountReceived);
        assertTrue(expectedUserBalanceAfterSellingToken !=
            actualUserBalanceAfterSellingToken);
    }
```

Recommended Mitigation:

Consider changing the implementation to use `TSwapPool::swapExactInput` instead. Note that this would require changing `sellPoolTokens` by adding additional parameter i.e., `minWethToReceive`.

```

```

```
    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));
    }
```

[H-04] `TSwapPool::_swap` gives users extra incentives, thereby breaking the protocol invariant `x * y = k`

Description: The protocol follows a strict invariant of `x * y = k` where, - `x`: Balance of pool token - `y`: Balance of WETH - `k`: Constant product of two balances

This means, whenever the balances change in the protocol, the ratio of balances should remain the same, hence the `k`. However, this is broken due to extra incentives given to users after `SWAP_COUNT_MAX` swaps. Meaning that, over time the protocol funds will be drained.

The following block of code is responsible for this 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 by doing a lot of swaps, collecting incentive given out by the protocol.

Proof of Concept:

1. A user swaps 10 times, collecting incentive of `1_000_000_000_000_000_000` tokens.
2. This user continues to swap, draining the protocol

PoC

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

```
7
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          int256 startingY = int256(weth.balanceOf(address(pool)));
24          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
26          vm.stopPrank();
27
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31
32          assertEq(expectedDeltaY, actualDeltaY);
33      }
```

Recommended Mitigation: Consider the following recommendations.

1. Remove the block of code providing the incentive.
2. If you want to keep providing the incentives, the protocol should account for the change in `x * y = k`.
3. Set aside the tokens the same way as we do in 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 -     }
```

Medium

[M-01] Missing Deadline checks in `TSwapPool::deposit()` causing transactions to complete even after the deadline

Description: The `deposit` function accepts deadline parameter which, according to documentation, "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 deposit rate is unfavorable.

Impact: Transactions could be sent when market conditions are unfavorable even when a deadline is added.

Proof of Concept: The `deadline` parameter is not used.

Recommended Mitigation: Consider making following changes to the function.

```
1
2       function deposit(
3           uint256 wethToDeposit,
4           uint256 minimumLiquidityTokensToMint,
5           uint256 maximumPoolTokensToDeposit,
6           uint64 deadline
7       )
8           external
9           revertIfZero(wethToDeposit)
10  +       revertIfDeadlinePassed(deadline)
11          returns (uint256 liquidityTokensToMint)
12      {
13
14          ...
15      }
```

Low

[L-01] `TSwapPool::LiquidityAdded` event in `_addLiquidityMintAndTransfer` function has parameters out of order

Description: When `LiquidityAdded` event is emitted in `TSwapPool::_addLiquidityMintAndTransfer` function, it logs values in the wrong order. The `poolTokensToDeposit` value should go as the third argument, whereas `wethToDeposit` should come as second argument.

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

Recommended Mitigation:

```
    function _addLiquidityMintAndTransfer(
        uint256 wethToDeposit,
        uint256 poolTokensToDeposit,
        uint256 liquidityTokensToMint
    ) private {
        _mint(msg.sender, liquidityTokensToMint);
-       emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+       emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit);

        ...
    }
```

[L-02] 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, it returns default value of zero, as named value is declared, but never assigned a value. This results in an incorrect indication of the amount.

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

Proof of Concept:

Recommended Mitigation:

```
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
    // reported @audit-gas should be external
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        // written @audit-low It's always going to return 0 instead of something else
-       returns (uint256 output)
+       returns (uint256 outputAmount)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));
```

```
18
19          uint256 outputAmount = getOutputAmountBasedOnInput(
20              inputAmount,
21              inputReserves,
22              outputReserves
23          );
24
25          if (outputAmount < minOutputAmount) {
26              revert TSwapPool__OutputTooLow(outputAmount,
                    minOutputAmount);
27          }
28
29          _swap(inputToken, inputAmount, outputToken, outputAmount);
30      }
```

Informationals

[I-01] Error `PoolFactory__PoolDoesNotExist` isn't used anywhere in `PoolFactory` and should be removed

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

[I-02] Lacking zero address check in constructor of `TSwapPool` and `PoolFactory` contracts.

```
1
2       constructor(address wethToken) {
3   +       if (wethToken == address(0)) revert();
4           i_wethToken = wethToken;
5       }
6
7       constructor(
8           address poolToken,
9           address wethToken,
10          string memory liquidityTokenName,
11          string memory liquidityTokenSymbol
12      ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
13  +       if (poolToken == address(0)) revert();
14  +       if (wethToken == address(0)) revert();
15          i_wethToken = IERC20(wethToken);
16          i_poolToken = IERC20(poolToken);
17      }
```

[I-03] `TSwapPool::createPool()` should use `.symbol()` instead of `.name()` for liquidityTokenSymbol

```
    function createPool(address tokenAddress) external returns (address
        ) {
        if (s_pools[tokenAddress] != address(0)) {
            revert PoolFactory__PoolAlreadyExists(tokenAddress);
        }
        string memory liquidityTokenName = string.concat("T-Swap ",
            IERC20(tokenAddress).name());
-       string memory liquidityTokenSymbol = string.concat("ts", IERC20
    (tokenAddress).name());
+       string memory liquidityTokenSymbol = string.concat("ts", IERC20
    (tokenAddress).symbol());

        ...
    }
```

[I-04] Event parameters should be indexed in `TSwapPool` contract.

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: 35

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

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
    event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
    event Swap(
```