T-Swap Audit Report

Version 1.0

Predator

November 4, 2024

T-Swap Audit Report

Predator

November 04, 2024

Prepared by: [Predator] Lead Security Researcher: - Predator

Table of Contents

    - [H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` break prootocol invariant of `x * y = k`

- Medium
    - [M-1] `TSwapPool::deposit` is missing `deadline` check causing transactions to complete even after the deadline
    - [M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariants

- Low
    - [L-1] `TSwapPool::LiquidityAdded` event has parametres out of order causing event to emit incorrect information
    - [L-2] Default value reurned by `TSwapPool::swapExactInput` results in incorrect return value given

- Informational
    - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
    - [I-2] Lacking zero address check
    - [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
    - [I-4] Event is missing `indexed` fields

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 PREDATOR 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

The findings are described in this document corresponf the following commit:

```
- 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

We spent around 3 hours with team to identify all vulnerabilities.

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	4
Gas Optimizations	0
Total	12

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes the protocol 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 miscalculated the resulting amount. When calculating the fee it's scales the amount by 10_000 instead of 1_000.

Impact Protocol takes more fees than expected from users.

Proof of Concepts

PoC

```
    function testSwapExactOutputTakesTooMuchTokens() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 200e18);
        poolToken.approve(address(pool), 200e18);
        pool.deposit(100e18, 0, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        address newUser = makeAddr("newUser");

        vm.startPrank(newUser);
        poolToken.mint(newUser, 11e18);
```

```
        // newUser buys 1 WETH from the pool with his poolToken
        poolToken.approve(address(pool), type(uint256).max);
        // After we swap, there will be ~110 tokenA, and ~91 WETH
        // 100 * 100 = 10,000
        // ~101 * 99 = 10,000
        pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
            timestamp));
        assertLt(poolToken.balanceOf(newUser), 1 ether);
        vm.stopPrank();
    }
```

Initially liquidity was 1:1, so user should get paid with 1 poolToken. However, he spent much more than that. The user started with 11 poolTokens, and now less than 1 poolToken left

Recommended mitigation

```
function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       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` causes user to potentially receive a way fewer tokens

Description The `swapExactOutput` function doesn't 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` should specify a `maxInputAmount`.

Impact If market conditions change before transaction processes, the user should get much worse swap

Proof of Concepts 1. The price of 1 WETH currently is 1.000 USDC. 2. User inputs a `swapExactAmount` looking for 1 WETH. 1. inputToken: USDC 2. outputToken: WETH 3. outputAmount: 1 WETH 4. deadline: type(uint64).max 3. The function doesn't offer a maxInput amount. 4. As the transaction is pending in the mempool, the market changes! And price moves HUGE! Now 1 WETH is 10.000 USDC. 10x more than

user expected. 5. The transaction completes, but the user sent to the protocol 10.000 USDC instead of 1.000 USDC.

Proof of code

```
        function testSwapExactOutputWithoutSlippageProtection() public {
        address newLp = makeAddr("newLp");
        vm.startPrank(newLp);
        weth.mint(newLp, 100e18);
        poolToken.mint(newLp, 100e18);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 0, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        address newUser = makeAddr("newUser");
        address newUser2 = makeAddr("newUser2");

        vm.startPrank(newUser);
        poolToken.mint(newUser, 10e18);
        poolToken.approve(address(pool), type(uint256).max);
        console.log("balance - 1 before", poolToken.balanceOf(newUser));
        console.log("balance - 1 before", weth.balanceOf(newUser));
        pool.swapExactOutput(poolToken, weth, 7 ether, uint64(block.timestamp + 1000));
        vm.stopPrank();

        vm.startPrank(newUser2);
        poolToken.mint(newUser2, 10e18);
        poolToken.approve(address(pool), type(uint256).max);
        console.log("balance - 2 before", poolToken.balanceOf(newUser2));
        console.log("balance - 2 before", weth.balanceOf(newUser2));
        pool.swapExactOutput(poolToken, weth, 7 ether, uint64(block.timestamp));
        vm.stopPrank();

        console.log("balance - 1 after", poolToken.balanceOf(newUser));
        console.log("balance - 1 after", weth.balanceOf(newUser));
        console.log("balance - 2 after", poolToken.balanceOf(newUser2));
        console.log("balance - 2 after", weth.balanceOf(newUser2));
    }
```

Recommended mitigation We should include `maxInputAmount`, so 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,
```

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

[H-3] `TSwapPoll::sellPoolTokens` miismatches input and output tokens causing users to receive 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 are willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculate the swaped amount.

This is due to the fact that the `swapExactOutput` fucntion is called, whereas the `swapExactOutput` 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, what is severe diruption of the protocol functionality.

Proof of Concepts

Recommended mitigation Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note, that it would also require the changing `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`).

```
1       function sellPoolTokens(
2   +       uint256 minEthToReceive,
3           uint256 poolTokenAmount
4           ) external returns (uint256 wethAmount) {
5   -       return swapExactOutput(i_poolToken, i_wethToken,
        poolTokenAmount, uint64(block.timestamp));
6   +       return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken
        , minEthToReceive, uint64(block.timestamp));
7       }
```

Additionaly, it might be wise to add a deadline to the function, as there is currently no deadline.

[H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` break prootocol invariant of `x * y = k`

Description The protocol follows a strict invariant of `x * y = k`. Where: - `x`: the balance of pool token; - `y`: the balance of weth token; - `k`: the constant product of the two balances

This means, whenever the balances chanhe in the protocol, the ratio between two tokens should be remain constant, hence the `k`. However, this is broken due to extra incentive in the `_swap` function. Meaning that over time the protocol 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 drain the protocol of funds by doing a lot of swaps and collecting the extra rewards given out by the protocol. More simply put, the protocol invariant is broken.

Proof of Concepts 1. User swaps 10 times and collects the extra incentives of `1_000_000_000_000_000_000` tokens, given out by the protocol. 2. That user continues to swap until drains the protocol of funds.

Proof of 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 outputWethAmount = 1e17;


        vm.startPrank(user);
        // Approve tokens so they can be pulled by the pool during the
           swap
        poolToken.approve(address(pool), type(uint256).max);

        // Execute swap, giving pool tokens, receiving WETH
        pool.swapExactOutput({
            inputToken: poolToken,
            outputToken: weth,
            outputAmount: outputWethAmount,
            deadline: uint64(block.timestamp)
        });
```

```
22          pool.swapExactOutput({
23              inputToken: poolToken,
24              outputToken: weth,
25              outputAmount: outputWethAmount,
26              deadline: uint64(block.timestamp)
27          });
28          pool.swapExactOutput({
29              inputToken: poolToken,
30              outputToken: weth,
31              outputAmount: outputWethAmount,
32              deadline: uint64(block.timestamp)
33          });
34          pool.swapExactOutput({
35              inputToken: poolToken,
36              outputToken: weth,
37              outputAmount: outputWethAmount,
38              deadline: uint64(block.timestamp)
39          });
40          pool.swapExactOutput({
41              inputToken: poolToken,
42              outputToken: weth,
43              outputAmount: outputWethAmount,
44              deadline: uint64(block.timestamp)
45          });
46          pool.swapExactOutput({
47              inputToken: poolToken,
48              outputToken: weth,
49              outputAmount: outputWethAmount,
50              deadline: uint64(block.timestamp)
51          });
52          pool.swapExactOutput({
53              inputToken: poolToken,
54              outputToken: weth,
55              outputAmount: outputWethAmount,
56              deadline: uint64(block.timestamp)
57          });
58          pool.swapExactOutput({
59              inputToken: poolToken,
60              outputToken: weth,
61              outputAmount: outputWethAmount,
62              deadline: uint64(block.timestamp)
63          });
64          pool.swapExactOutput({
65              inputToken: poolToken,
66              outputToken: weth,
67              outputAmount: outputWethAmount,
68              deadline: uint64(block.timestamp)
69          });
70          pool.swapExactOutput({
71              inputToken: poolToken,
72              outputToken: weth,
```

```
            outputAmount: outputWethAmount,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

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

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);

        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended mitigation Remove the extra incentives mechanism. If you want to keep it 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-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 to the documentation, is "The deadline for the transaction to be completed by". However, this paramenter is never used. As a consequence, operations adding a liquidity might be executed at unexpected times, in market conditions where the deposit rate is unfavourable.

Impact Transactions might be sent when conditions are unfavourable to deposit, even when adding a deadline parameter.

Proof of Concepts 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
        revertIfZero(wethToDeposit)
+       revertIfDeadlinePassed(deadline)
        returns (uint256 liquidityTokensToMint)
    {
```

[M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariants

Description The core invariant of the protocol is: `x * y = k`. In practice though, the protocol takes fees and actually increases k. So we need to make sure `x * y = k` before fees are applied

Recommended mitigation Protocols must be explicitly designed to handle these edge cases. For example: - Adding logic to handle rebases by periodically recalculating balances. - Implementing fee-aware transfer functions to account for tokens with fees. - Being cautious with ERC-777 tokens by either disabling hooks or ensuring reentrancy-safe code.

Low

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

Description When `LiquidityAdded` event is emtted in the `TSwapPool::_addLiquidityMintAndTransfer` function, it logs value in incorrect order. The `poolTokensToDeposit` value should go in the third parameter section, whereas the `wethToDeposit` value should go second.

Impact Event emssion 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 reurned by `TSwapPool::swapExactInput` results in incorrect return value given

Description The `swapExactInput` function is expected to return the actual amount of token bought by the caller. However, while it declares the named the return value `output` it is never assigned as a

value, nor uses an explicit return statement.

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

Proof of Concepts Launch the following code in to `TSwapPool.t.sol` from test/unit folder:

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

        vm.startPrank(user);
        poolToken.approve(address(pool), 10e18);
        uint256 outputAmount = pool.swapExactInput(poolToken, 10e18,
            weth, 0, uint64(block.timestamp));
        vm.stopPrank();
        assertEq(outputAmount, 0);
    }
```

Recommended mitigation

```
     function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount, // e minimum amount of output tokens
            to receive
        uint64 deadline //e deadline for the transaction to be
            completed by
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
-       returns (uint256 output)
+       returns (uint256 outputAmount)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-       uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
     inputReserves, outputReserves);
+       outputAmount = getOutputAmountBasedOnInput(inputAmount,
    inputReserves, outputReserves);
```

Informational

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

```
    contract PoolFactory {
        error PoolFactory__PoolAlreadyExists(address tokenAddress);
-       error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

[I-2] Lacking zero address check

```
+   event PoolFactory__ZeroAddress(address zeroAddress)
    constructor(address wethToken) {
+           if (wethToken == address(0)) {
+               revert PoolFactory__ZeroAddress(wethToken);
+}

            i_wethToken = wethToken;
    }
```

Note the same problem in `constructor` of `TSwapPool` for `address address poolToken` & `address 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());
```

[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.

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(
```

**