Tswap Protocol Audit Report

Prepared by: 0xshuayb

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).

Disclaimer

The 0xshuayb 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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

Scope

- In Scope:

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

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
    - Any ERC20 token

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.

Issues found

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

Findings

High

[H-1] Incorrect fees 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 the amount of tokens a user should deposit given an amount of output tokens. However, the function miscalculates the resulting

amount. When calculating the fees, it scales the amount by 10_000 instead of 1_000.

Impact Protocol takes more fees than expected from users

Proof of Concepts

▶ PoC

```
 function testGetInputAmountBasedOnOutput() 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 outputAmount = 10 * 1e18;
        uint256 inputReserves = poolToken.balanceOf(address(pool));
        uint256 outputReserves = weth.balanceOf(address(pool));

        uint256 expectedInput = ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
        // int256 newExpectedInput = int256(expectedInput);

        uint256 computedInput =
(pool.getInputAmountBasedOnOutput(outputAmount, inputReserves,
outputReserves));

        assertEq(computedInput, expectedInput);
    }
```

Running the test above shows a diffrence in the `computedAmount` and the `expectedAmount` which is due to the error in the scaling factor

Recommended mitigation

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-         return ((inputReserves * outputAmount) * 10000) /
((outputReserves - outputAmount) * 997);
+         return ((inputReserves * outputAmount) * 1000) / ((outputReserves
```

```
- outputAmount) * 997);
    }
```

[H-2] Lack of slippage protection in `TswapPool::swapExactOutput` causesusers to potentially receive way fewer tokens than expected

Description The `swapExactOutput` function does not include any form of slippage protection. This function is similar to what was done in `TswapPool::swapExactInput`, where the function specifies a `minOutputAmount`. The `swapExactOutput` should specify a `maxInputAmount`

Impact if market condition changes before the transaction goes through, users could get a bad swap

1. The price of 1 WETH right now is 1_000 USDC
2. User inputs a `swapExactOutput` looking for 1 WETH.
    1. inputToken = USDC
    2. outputToken = WETH
    3. outputAmount = 1 WETH
    4. deadline = 0
3. The function does not offer a maxInputAmount
4. As the transaction is pending in the memepool, the maeket changes! 1 WETH --> 10_000 USDC. 10x more than what the user anticipated
5. The transaction completes but the user sent the protocol 10_000 USDC instead of 1_000 USDC for 1 WETH

Recommended mitigation We should include a `maxInputAmount` so the user only has to spend up to a specified amount, the maximun amount they are willing to spend on that particular swap

```
    function swapExactOutput(
        IERC20 inputToken,
+        unt256 maxInputAmount,
.
.
.
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

        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 token and receive WETH in exchange. Users indicate how many pool tokens they are willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculates the swapped amount. Thus 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 the protocol's functionality

Recommended mitigation Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would require changing the `sellPoolTokens` function to accept a new parameter (i.e. `minEthToReceive` be passed to `swapExactInput`)

```
 function sellPoolTokens(
        uint256 poolTokenAmount,
+        uint256 minEthToReceive
    ) external returns (uint256 wethAmount) {
-        return swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount,
uint64(block.timestamp));
+        return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken,
minEthToReceive, uint64(block.timestamp));
    }
```

Additionally, it is avised to add a deadline to the function as there is'nt any currently

[H-4] In `TswapPool::_swap`the extra tokens given to the user after every `swapCount` breaks the invariant protocol 0f `x * y = k`

Description The protocol follows a strict invariant of `x * y = k`. Where:

- `x`: The balance of he 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 of the two amounts must remain constant. Howwever, this is broken due to the extra incentive in the `_swap` function. This implies that over time, the protocol's funds will be drained. The following 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. The protocol's core invariant is broken

Proof of Concepts

1. A user swap 10 times and collects the extra incentive of 1_000_000_000_000_000_000 tokens
2. That user continues to swap until all protocol funds are drained

▶ PoC

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

        vm.startPrank(user);
        poolToken.approve(address(pool),type(uint256).max);
        poolToken.mint(user, 100e18);
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));

        int256 actualDeltaY = int256(endingY) - int256(actualStartingY);
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended mitigation Remove the extra incentive mechanism. If you want to keep this, you should account for the change in `x * y = k` or, we should set aside tokens in the same way we do for 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 go through even when the dealine has passed

Description The `deposit` function accepts a dealine parameter which according to the documentation is "The deadline for the transaction to be completed by". However, the parameter was never used. This could cause deposits at unexpected times, in market condition where the deposit rate is unfavourable.

Impact Transactions could be sent when market condition is unfavourable even when a deadline parameter is added

Proof of Concepts The `dealine` parameter is unused.

Recommended mitigation

```
    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 tokens break the protocol invariant

Description When a user makes 10 swaps using the protocol, the protocol's fee-on-transfer mechanism rewards the user with extra incentive as seen in `TswapPool::_swap`

```
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. The protocol's core invariant is broken

Proof of Concepts

1. A user swap 10 times and collects the extra incentive of 1_000_000_000_000_000_000 tokens
2. That user continues to swap until all protocol funds are drained

▶ PoC

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

        vm.startPrank(user);
        poolToken.approve(address(pool),type(uint256).max);
        poolToken.mint(user, 100e18);
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));

        int256 actualDeltaY = int256(endingY) - int256(actualStartingY);
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended mitigation Remove the extra incentive mechanism. If you want to keep this, you should account for the change in `x * y = k` or, we should set aside tokens in the same way we do for fees

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

Low

[L-1] `TswapPool::LiquidityAdded` event has paramters out of order causing the event to emit incorrect information

Description When the `LiquidityAdded` event is emitted in the `TswapPool::_addLiquidityMintAndTransfer` function, it logs values in an incorrect order. The `poolTokenToDeposit`value should go in the third position while `wethToDeposit` comes second

Impact Event emission is incorrect, leading to potential off-chain malfunctions

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 `swapExactOutput` function is expected to the actual amount of tokens bought by the caller. however, while the return value is named `output`, it is never assigned a value

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

Recommended mitigation

```
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

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

        if (outputAmount < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
```

```
        }

        _swap(inputToken, inputAmount, outputToken, outputAmount);
    }
```

Informational

[I-1] `PoolFctory::PoolFactory__PoolDoesNotExist` is not used at all in the contract and should be removed

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

[I-2] Lacking zero address check

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

[I-3] `PoolFctory::createPool` should use `.symbol()` instead on `.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` field.

```
-    event LiquidityAdded(
-        address indexed liquidityProvider,
-        uint256 wethDeposited,
-        uint256 poolTokensDeposited
-    );
+    event LiquidityAdded(
+        address indexed liquidityProvider,
+        uint256 indexed wethDeposited,
+        uint256 indexed poolTokensDeposited
+    );
-    event LiquidityRemoved(
-        address indexed liquidityProvider,
-        uint256 wethWithdrawn,
-        uint256 poolTokensWithdrawn
-    );
+    event LiquidityRemoved(
```

```
+        address indexed liquidityProvider,
+        uint256 indexed wethWithdrawn,
+       uint256 indexed poolTokensWithdrawn
+    );
-    event Swap(
-        address indexed swapper,
-        IERC20 tokenIn,
-        uint256 amountTokenIn,
-        IERC20 tokenOut,
-        uint256 amountTokenOut
-    );

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