TSwap Audit Report

Version 1.0

nem0x001

May 23, 2024

TSwap Audit Report

nem0x001

March 23, 2024

Prepared by: nem0x001

Table of Contents

- Medium
  * [M-1] DeadLine is not used.causing transactions to complete after the deadline.
- Low
  * [L-1] `TswapPool::LiquidityAdded` incorrect order leading to false information .
  * [L-2] The `TswapPool::swapExactInput` function return value is not correct
- Informational
  * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used any where
  * [I-2] Lacking Zero Address Check.
  * [I-3] Liquidity token symbol naming problem.
  * [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

nem0x001 has made 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

		Impact		
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

This Audit report for the source code of CommitHash `e643a8d4c2c802490976b538dd009b351b1c8dda` of Tswap protocol.

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

Audit Duration The audit of the Tswap protocol was conducted over a period of 8 days.

Audit Methodology The audit was performed using a comprehensive approach that included the following methods:

1. **Manual Review**
   - A thorough manual examination of the smart contract code was performed to identify potential security vulnerabilities, logical errors, and areas of improvement. This involved reviewing the code line-by-line to ensure correctness and adherence to best practices.
2. **Unit Testing**
   - Extensive unit tests were written and executed to validate the functionality of individual components of the protocol. These tests aimed to cover a wide range of scenarios to ensure each function behaves as expected under various conditions.
3. **Invariant Testing**
   - Invariant testing was employed to check the consistency and stability of the protocol. This involved creating tests that assert the correctness of certain properties or conditions that should always hold true, regardless of the state or inputs of the protocol.

The combination of these methods provided a robust and thorough assessment of the Tswap protocol, ensuring that it meets the highest standards of security and functionality.

Issues found

Severity	Number of Issues found
High	3
Medium	1
Low	2
Info	4
Total:	10

Findings

High

[H-1] Incorrect fee calculations in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take a large amount of tokens from users,as fees

Description:

The `TSwapPool::getInputAmountBasedOnOutput` function calculate the amount of tokens the user should deposit equivalent to the amount of output tokens.however the function didn't calculate fees correctle as it scale amount by 10_000 instead of 1_000.

Impact:

Protocol take alot of tokens from user as fees.

Proof of Concept:

Use the following code in your test.

```
function testgetInputAmountBasedOnOutputAmountWrongFees() public {
      //first deposit LQ
      vm.startPrank(liquidityProvider);
      weth.approve(address(pool), 100e18);
      poolToken.approve(address(pool), 100e18);
      pool.deposit(100e18, 0, 100e18, uint64(block.timestamp));
      vm.stopPrank();
      // here i want to get i will
      vm.startPrank(user);
      poolToken.approve(address(pool), 100e18);
      //pool balance : 100 weth , 100 poolToken
      // user want to swap pool token to get 1 weth
      // 100 * 100 = 10,000
      // 101 * 99 = 9,999
      console.log("weth balance of user", weth.balanceOf(user));
      console.log("poolToken balance of user", poolToken.balanceOf(user));
      pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.timestamp));

      console.log("weth balance of user", weth.balanceOf(user));
      console.log("poolToken balance of user", poolToken.balanceOf(user));

      assertLt(poolToken.balanceOf(user), 1 ether);
  }
```

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] The `TSwapPool::swapExactOutput` has no slippage protection.which lead to worse swapping prices.

Description:

The `swapExactOutput` function has no slippage protection.This function is simmilar to `TSwapPool::swapExactInput` where it specify `minOuputAmount`. the `swapExactOutput` should specify a `maxInputAmount`

Impact:

if the market condition changes while the transaction is processing.this will lead to worse swap prices.

Proof of Concept:

1. The price of 1 WETH right now is 1,000 USDC
2. User inputs a swapExactOutput looking for 1 WETH
   - inputToken = USDC
   - outputToken = WETH
   - outputAmount = 1
   - deadline = whatever
3. The function does not offer a maxInput amount
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

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,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
+         uint maxInputAmount
```

```
    )
        public
        revertIfZero(outputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 inputAmount)
    {
        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 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 miscalculaes 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 wrong amount of tokens.which disturb one of the protocol main functions.

Recommended Mitigation:

Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new parameter (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));
      }
```

[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 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.

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 untill all the protocol funds are drained

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

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);
        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));

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

        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended Mitigation:

1. Change the protocol invariant `x*y=k` .
2. remove the incentive

Medium

[M-1] DeadLine is not used.causing transactions to complete after the deadline.

Description:

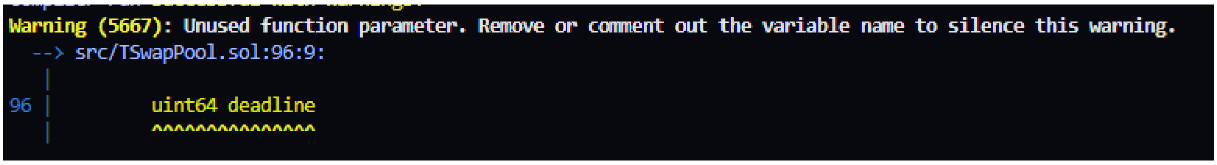
The `TSwapPool::deadline` param is not used anywhere in the `TSwapPool::deposit()` fucntion however according to the documentation this param is reponsible for The deadline for the transaction to be completed by.

Impact:

Transactions can be completed after the deadline has passed.

Proof of Concept:

According to the compiler output `deadLine` param is not used.



```
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> src/TSwapPool.sol:96:9:
   |
96 |         uint64 deadline
   |         ^^^^^^^^^^^^^^^
```

Recommended Mitigation:

we can use `TSwapPool::revertIfDeadlinePassed` modifier as following.

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
        revertIfZero(wethToDeposit),
+        revertIfDeadlinePassed(deadline)
        returns (uint256 liquidityTokensToMint)
```

Low

[L-1] `TswapPool::LiquidityAdded` incorrect order leading to false information .

Description:

The `TswapPool::LiquidityAdded` event track the `Lq provider` , `wethDeposited` and `poolToken Deposited` but when it used in `TswapPool::_addLiquidityMintAndTransfer` function it emited with wrong order

Impact:

Tracking of false info

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-2] The `TswapPool::swapExactInput` function return value is not correct

Description:

The `TswapPool::swapExactInput` should return the ouput token amount that will be sent to the user.since the ouput is not used or intiallized in the function so the return value will always be zero.

Impact:

False amount will be shown to the caller of this function.

Proof of Concept:

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

        vm.startPrank(user);
        poolToken.approve(address(pool), 10e18);
        uint256 returnValue = pool.swapExactInput(poolToken, 10e18,
            weth, 1e18, uint64(block.timestamp));

        assertEq(returnValue, 0);
    }
```

Recommended Mitigation:

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {
        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);
        }

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

Informational

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used any where

Description:

The error `PoolFactory__PoolDoesNotExist` is not used in the `PoolFactory` contract.

Impact:

Higher Gas Prices.

Recommended Mitigation:

Remove the `PoolFactory__PoolDoesNotExist` error from the code.

[I-2] Lacking Zero Address Check.

Description:

`PoolFactory::constructor` address passed to the constructor there is no check to make sure it's a valid one.

Recommended Mitigation:

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

 i_wethToken = wethToken;
```

[I-3] Liquidity token symbol naming problem.

Description:

The symbol name of `PoolToken::liquidityTokenSymbol` is not named correctly.since the code is concatinating the `ts` with the `IERC20.name()` not `IERC20.symbol()`

Recommended Mitigation:

```
-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: 35

  ```
      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);
```