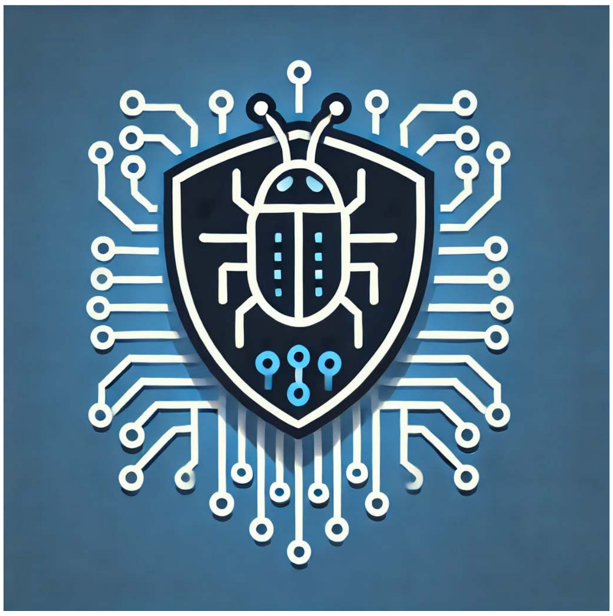

TSwap Protocol Audit Report

Version 1.0

Cyfrin.io

November 7, 2024

TSwap Protocol Audit Report

Ivan Kartunov

November 7, 2024

Prepared by: Ivan Kartunov Lead Auditors: - Ivan Kartunov

Table of Contents

- Medium
  * [M-1] `TSwapPool::deposit` is missing deadline check causing transaction to complete even after
  * [M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant
- Low
  * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emit incorrect information
  * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given.
- Infomationals
  * [I-1] `PoolFactory::PoolFactory__PoolDOesNotExist` is not used and shoudl be removed
  * [I-2] Lacking zero address checks
  * [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
  * [I-4] Event is missing `indexed` fields

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Ivan Kartunov's 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

		Impact		
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

Scope

Roles

Executive Summary

Issues found

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

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in a loss of funds

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens the user should deposit given an amount of tokens of output tokens. However, the function currently misscaulculates the resulting amount. When calculating the fee, it scales the amount by 10_000 insted of 1_000.

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

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 users to potentially receive less tokens than expected

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 `maxOutputAmount`

Impact: If market contitions change before the transaction procceses, the use could get much worse swap.

Proof of Concept: 1. The price now of 1 WETH is 1_000 USDC 2. User Inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = …

3. The function does not offer maxInput Amount 4. As the transaction is pending in the mempool, the market changes and the price moves to HUGE -> 1 WETH is now 10_000 USDC. 10x more than the user expected. 5. The transaction is processed and the user receives 1 WETH, but the price is now 10_000 USDC, so the user has lost 9_000 USDC.

Recommended Mitigation: We should include a `maxInputAmount` parameter in the function signature, so the user only has to spend up to 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 incorrect amount tokens

Description: The `sellPoolTokens` function is intended to allow users to sell their pool tokens for WETH. Users indicate how many pool tokens they are 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 the users specify the exact amount of input tokens not output tokens.

Impact: Users will swap the wrong amount of tokens, potentially losing funds.

Recommended Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this 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));
  }
```

Additonally, it might be wise to add a deadline to the function, as there is no deadline check in the function.

[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 strict invariant of `x * y = k`. Where: - `x`: The balance of the pool token - `y`: The balance of the WETH token - `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++;
    if (swap_count >= SWAP_COUNT_MAX) {
        swap_count = 0;
        outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000)
            ;
    }
```

Impact: A user could potentially drain the protocol of all funds by doing a lot of swaps and collecting the extra incentive given out by the protocol.

Proof of Concept: 1. A user does a lot of swaps, and collects the extra incentive given out by the protocol. 2. That user continues to do swaps until all the protocol funds are drained.

Proof Of Code

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

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

uin256 outputWeth = 1e17;

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

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

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

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

Recommended Mitigation: Remove the extra incentive. If you want to keep this in, we should account for the change in the x * y = k invariant.

```
-     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 transaction to complete even after

the deadline

Description: The `deposit` accepts a `deadline` parameter. However this parameter is never used. As a consequence, operations that add liquidity to the pool can be executed at unexpected times,in a market conditions where the deposit rate is unfavorable.

Impact: Transaction 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 ERC777 tokens break protocol invariant

Low

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

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

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 user. However, while it declares the named return value `output` it is never assigned a value, nor uses an explict return statement.

Impact: The return value always defaults to 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 (output < minOutputAmount) {
-        revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
    }
+    if (output < minOutputAmount) {
+        revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
    }

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

Infomationals

[I-1] `PoolFactory::PoolFactory__PoolDOesNotExist` is not used and shoudl be removed

```
- error PoolFactory::PoolFactory__PoolDOesNotExist(address tokenAddress);
```

[I-2] Lacking zero address checks

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

- 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);
  ```