T-Swap Protocol Audit Report

Version 1.0

Cyfrin.io

March 19, 2024

T-Swap Audit Report

L.Petroulakis

March 12, 2024

Prepared by: [L.Petroulakis] Lead Auditors: - L.Petroulakis

Table of Contents

- Medium severity findings
  - [M-1] `TSwapPool::deposit` is missing the deadline check it is supposed to have, causing transactions to complete even after it.
- Low severity findings
  - [L-1] `TSwapPool::LiquidityAdded` event has inverted parameters
  - [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informational Findings
  - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  - [I-2] Lack of zero address checks
  - [I-3] `PoolFactory::createPool` should should be using `.symbol` instead of `.name()`
  - [I-4] Event is missing `indexed` fields

Protocol Summary

None needed.

Disclaimer

None needed.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

Scope

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	4
Total	9

Findings

High severity findings

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

Description The `getInputAmountBasedOnOutput` function calculates the amount of tokens to be deposited by the user based on the amount of tokens they want to receive. However, the function currently miscalculates the resulting amount as when it calculates fees, it scales the amount by 10,000 instead of 1,000.

Impact More fees are taken from the user than intended.

Recommended mitigation Consider changing the following.

```
1     function getOutputAmountBasedOnInput(
2         uint256 inputAmount,
3         uint256 inputReserves,
4         uint256 outputReserves
```

```
    )
        public
        pure
        revertIfZero(inputAmount)
        revertIfZero(outputReserves)
        returns (uint256 outputAmount)
    {

        uint256 inputAmountMinusFee = inputAmount * 997;
        uint256 numerator = inputAmountMinusFee * outputReserves;
        uint256 denominator = (inputReserves * 1000) +
           inputAmountMinusFee;
        return numerator / denominator;
    }

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

[H-2] No Slippage Protection in `TSwapPool::swapExactOutput` causes users to potentially receive much fewer tokens than expected.

Description The `swapExactOutput` function does not have any kind of slippage protection. It is similar to what is dome in `TSwapPool::swapExactInput` but in the opposite direction. The `swapExactOutput` function should specify a `maxInputAmount`.

Impact If market conditions change before the completion of the transaction, users can receive much fewer tokens than expected.

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
4. 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 and 1 WETH is now worth 10,000 USDC, which is 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,
+       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 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 the wrong amount of tokens, which is a severe disruption of protcol functionality.

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));
    }
```

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline. (MEV later)

[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 follow 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 maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol.

Most simply put, the protocol's core invariant is broken.

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 until all protocol funds are drained

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: Remove the extra incentive mechanism. If you want to keep this 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++;
-        // Fee-on-transfer
-        if (swap_count >= SWAP_COUNT_MAX) {
-            swap_count = 0;
-            outputToken.safeTransfer(msg.sender, 1
```

```
        _000_000_000_000_000_000);
-       }
```

Medium severity findings

[M-1] `TSwapPool::deposit` is missing the deadline check it is supposed to have, causing transactions to complete even after it.

Description The `deposit` function accepts a `deadline` parameter which according to the documentation is the deadline for the transaction to be completed. However, the parameter is never used and operations that add liquidity the pool can be executed at any time. This makes our protocol susceptible to MEV attacks.

Impact Transactions can be sent when market is unfavorable to deposit.

Proof of Concept: The `deadline` parameter is unused.

Recommended mitigation Consider adding the following.

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

Low severity findings

[L-1] `TSwapPool::LiquidityAdded` event has inverted parameters

Description When the event is emitted in the `TSwapPool::addLiquidityMintAndTransfer` function, the values are logged incorrectly.

Impact Event emission is incorrect.

Recommended mitigation

```
- emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDepost);
+ emit LiquidityAdded(msg.sender, wethToDepost, 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 token received. However, the `output` is never assigned a value.

Recommended mitigation

```
{ 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);
+ if (output < minOutputAmount) {
+ revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
}
-    _swap(inputToken, outputToken, inputAmount, outputAmount);
+   _swap(inputToken, outputToken, inputAmount, output);}
```

Informational Findings

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

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

[I-2] Lack of zero address checks

```
constructor(address wethToken) {
+    if(wethToken == address(0)) {
+        revert("WETH token address cannot be zero");
+    }
    i_wethToken = wethToken;
}
```

[I-3] `PoolFactory::createPool` should should be using `.symbol` instead of `.name()`

```
-   string memory liquidityTokenSymbol = string.concat("ts", IERC20(tokenAddress).name());
```

```
2 +    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

```
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(
```