Protocol Audit Report

Version 1.0

Cyfrin.io

December 3, 2025

T-Swap Audit Report

Mayank

December 3, 2025

Prepared by: [Mayank] Lead Auditors: - Mayank Mokta

Table of Contents

Protocol Summary

This is a project kind of similar to UniswapV1 where : 1. Any user can deposit liquidity tokens and can get profit in return. 2. Any user can swap tokens from weth to poolToken or vice-versa. 3. User can sell input tokens as well and get the output tokens in return based on the price.

Disclaimer

The Mayank Mokta 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 me is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solidly 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

Below we have our commit hash

```
a6ca88657fba670715364fc77ee5c1cc1984b3a9
```

Scope

```
./src/TSwapPool.sol
./src/PoolFactory..sol
```

Roles

- Liquidator: The person who can deposit liquidity tokens and get profit and liquidity tokens in return.
- User: An user who can swap tokens.

Executive Summary

Issues found

Severity	Number of issues found
HIGH	4
MEDIUM	1
LOW	2
INFO	11
TOTAL	18

Findings

High

[H-1] TITLE (Root Cause + Impact) In `TSwapPool::_swap` user gets 1`_000_000_000_000_000_000` output tokens for every 10 transactions which breaks the protocol of `x * y = k`.

Description: When user wants to swap the exact input or exact output, the user call either `swapExactOutput` or `swapExactInput` of in these functions the function `TSwapPool::_swap` gets called which pays the user 1`_000_000_000_000_000_000` output tokens for every 10 transactions which breaks the protocol of `x * y = k`. Since we want the protocol to follow `x * y = k` after every swap it breaks when user completes his 10 transactions.

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

Impact: The main protocol of the contract get broken i.e `x * y = k` and any attacker could even swap a lot of times draining the funds of contract very easily

Proof of Concept: Consider adding the below test in your `TSwapPoolTest`

PoC

```
function testSwapBreaksContractProtocol() external{
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), type(uint256).max);
    poolToken.approve(address(pool), type(uint256).max);
    pool.deposit(100e18, 10e18, 100e18, uint64(block.timestamp));
    vm.stopPrank();
    vm.startPrank(user);
    uint256 startingUserPoolTokenBalance = poolToken.balanceOf(address(
        user));
    weth.approve(address(pool),type(uint64).max);
    uint256 wethInputReserves = weth.balanceOf(address(pool));
    uint256 poolTokenOutputReserves = poolToken.balanceOf(address(pool)
        );
    uint256 oneOutputPoolTokenAmount = pool.getOutputAmountBasedOnInput
        (1e18, wethInputReserves, poolTokenOutputReserves);
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    pool.swapExactInput(weth, 1e18, poolToken, 0, uint64(block.
        timestamp));
    uint256 endingUserBalance = poolToken.balanceOf(address(user));
    uint256 expectedUserPoolTokenBalance = startingUserPoolTokenBalance
         + (oneOutputPoolTokenAmount * 11) ;
    assert(endingUserBalance > expectedUserPoolTokenBalance);
    vm.stopPrank();
}
```

Recommended Mitigation: You are recommended to remove this reward logic of giving free `1_000_000_000_000_000_000` output tokens to the user. Consider adding following changes to the `TSwapPool::_swap` function.

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

[H-2] TITLE (Root Cause + Impact) In `TSwapPool::sellPoolTokens` we are actually swapping `wethAmount` but we want `wethAmount` which is totally opposite.

Description: In `TSwapPool::sellPoolTokens` we actually want to sell out `poolTokens` and in return get `weth` so in order to do this we should call `swapExactInput` because our input is our parameter i.e `uint256 poolTokenAmount`, but instead we are calling `swapExactOutput` which will consider `poolTokens` as output amount and `weth` as input.

Impact: User will get `poolTokens` instead of `weth`.

Proof of Concept: Consider adding the below test in your `TSwapPoolTest`

PoC

```
function testSellPoolTokensReturnsPoolTokens() external {
    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);
    uint256 startingWethBalance = weth.balanceOf(address(user));
    uint256 startingPoolTokenBalance = poolToken.balanceOf(address(user
       ));
    poolToken.approve(address(pool),type(uint256).max);
    weth.approve(address(pool),type(uint256).max);
    pool.sellPoolTokens(1e18);
    uint256 endingWethBalance = weth.balanceOf(address(user));
    uint256 endingPoolTokenBalance = poolToken.balanceOf(address(user))
       ;
    assert(startingPoolTokenBalance > endingPoolTokenBalance);
    vm.stopPrank();
}
```

Recommended Mitigation: Consider adding the below code in your `TSwapPool::sellPoolTokens`

```
-     function sellPoolTokens(uint256 poolTokenAmount) external returns
    (uint256 wethAmount) {
```

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

[H-3] TITLE (Root Cause + Impact) Incorrect fee calculation in the function `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take way too many tokens from user.

Description: In function `TSwapPool::getInputAmountBasedOnOutput` the calculation is totally wrong in my opinion because you want your protocol to give 0.03% of every swap to the liquidators but this function will actually returns wrong value of input based of output.

Impact: The protocol takes way too many fees from the user

Proof of Concept: COnsider adding the below test to your `TSwapPoolTest`

PoC

```
function testGetInputAmountBasedOnOutputChargesAlot() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool),100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        uint256 inputReservesWeth = weth.balanceOf(address(pool));
        uint256 outputReservesPToken = poolToken.balanceOf(address(pool
            ));
        uint256 expectedInputWeth = 11144544745347152568;
        uint256 actualInputWeth = pool.getInputAmountBasedOnOutput(10
            e18, inputReservesWeth, outputReservesPToken);
        console.log("the actual input amount is as of 10e18Weth is:",
            actualInputWeth);
        assert(actualInputWeth > expectedInputWeth);
        vm.stopPrank();
    }
```

Recommended Mitigation: Consider adding the below code in `TSwapPool::getInputAmountBasedOnOutput` function.

```
return
-             ((inputReserves * outputAmount) * 10000) /
+             ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
```

[H-4] TITLE (Root Cause + Impact) Lack of slippage protection in `TSwapPool::swapExactOutput` function.

Description: The `TSwapPool::swapExactOutput` does not provide any sort of slippage protection. First, there should definitely be a parameter saying `uint256 minInputAmount` and then there should be a check for minInputAmount as well if it exceeds the actual input amount, which is miind of similar to what you did in the function `TSwapPool::swapExactInput`.

Impact: If market conditions drops somehow, user will get a much worse swap.

Proof of Concept: 1. Suppose the price of 1WETH is 100 USDC. 2. User sees this he wants 1WETH and calls `swapExactOutput` with parameters as (USDC,WETH,1,block.timestamp) 3. Suddenly the price rises to 1WETH is 200USDC 4. S the user pays 200USDC to get 1WETH, 2X more the amound user paid.

Recommended Mitigation: Consider adding the below code to the `TSwapPool::swapExactOutput` function.

```
-    function swapExactOutput(IERC20 inputToken,IERC20 outputToken,
    uint256 outputAmount,
+    function swapExactOutput(IERC20 inputToken,IERC20 outputToken,
    uint256 outputAmount,uint256     minInputAMount
     uint64 deadline
     )
         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(minInputAMount > inputAmount){
+        revert();
 }
         _swap(inputToken, inputAmount, outputToken, outputAmount);
     }
```

Medium

[M-1] TITLE (Root Cause + Impact) In `TSwapPool::deposit` the deadline check for the transaction is missing.

Description: the function `TSwapPool::deposit` has a parameter `uint64 deadline` which is not used anywhere plus according to your natspac this parameter should ensure that no deposit transaction will continue if the deadline has passed. However the paramaeter is not used or updated so anytime the deposit can happen even after the transaction deadline as well which can even lead to `MEV attack`

Impact: Transactions can be send when market conditions are unfavourable to deposit, even when adding a deadline parameter.

Proof of Concept: The `uint64 deadline` parameter is unused.

Recommended Mitigation: Consider making following changes to the function `TSwapPool::deposit`:

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

Low

[L-1] TITLE (Root Cause + Impact) In `TSwapPool::_addLiquidityMintAndTransfer` function the event emitted is backwords which is incorrect

Description: In the function `TSwapPool::_addLiquidityMintAndTransfer` there is emitted event `LiquidityAdded` in which the order of events emitted is incorrect as the second one should be on the third place and the third one should be on the second place.

Impact: This could lead to getting wrong values off-chain or on frontend if we call this event or get data.

Recommended Mitigation: Consider adding this code to your `TSwapPool::_addLiquidityMintAndTransfer` function:

```
-    emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit
    );
+    emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit
    );
```

[L-2] TITLE (Root Cause + Impact) Incorrect return value is been returned in `TSwapPool::swapExactInput`

Description: the function `TSwapPool::swapExactInput` returns `uint256 output`, but the returns is actually never used anywhere in the function, which is incorrect its name is output so it might return the output value.

Impact: The returned value will always be 0, which is wrong information

Proof of Concept: `uint256 output` is never used in the function `TSwapPool::swapExactInput`

Recommended Mitigation: Consider adding the below code to your function `TSwapPool::swapExactInput`:

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

Informational

[I-1] TITLE (Root Cause + Impact) The error `PoolFactory::PoolFactory__PoolDoesNotExist` is not used in the contract, hence should be removed.

```

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

[I-2] TITLE (Root Cause + Impact) Lacking zero address check at constructor in `PoolFactory`

```

constructor(address wethToken) {
+        require(wethToken != address(0), "Weth Address is zero")
```

```
            i_wethToken = wethToken;
        }
```

[I-3] TITLE (Root Cause + Impact) In `PoolFactory::createPool` .symbol can be used instead of .name

```

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

[I-4] TITLE (Root Cause + Impact) `TSwapPool::Swap` indexed can be used in this event

```
-         IERC20 tokenIn,
+         IERC20 indexed tokenIn,
        uint256 amountTokenIn,
-         IERC20 tokenOut,
+         IERC20 indexed tokenOut,
```

[I-5] TITLE (Root Cause + Impact) In `TSwapPool::deposit` variable not used anywhere hence, can be removed

```
-     uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```

[I-6] TITLE (Root Cause + Impact) In `TSwapPool::deposit` CEI not being followed

```
else {
+           liquidityTokensToMint = wethToDeposit;
            _addLiquidityMintAndTransfer(
                wethToDeposit,
                maximumPoolTokensToDeposit,
                wethToDeposit
            );
-            liquidityTokensToMint = wethToDeposit;
        }
```

[I-7] TITLE (Root Cause + Impact) In `TSwapPool::getInputAmountBasedOnOutput` constants should be used instead of magic numbers

```

+         uint256 public constant TOTAL_PERCENT = 10000;
+         uint256 public constant MARGIN = 997;
-         ((inputReserves * outputAmount) * 10000) /
-         ((outputReserves - outputAmount) * 997);
+         ((inputReserves * outputAmount) * TOTAL_PERCENT) /
+         ((outputReserves - outputAmount) * MARGIN);
```

[I-8] TITLE (Root Cause + Impact) `TSwapPool::swapExactInput` should be marked as `external` instead of public

```

function swapExactInput(
        IERC20 inputToken,uint256 inputAmount,IERC20 outputToken,
           uint256 minOutputAmount,uint64 deadline
    )
-         public
+         external
```

[I-9] TITLE (Root Cause + Impact) `TSwapPool::totalLiquidityTokenSupply` can be marked as `external` instead of public

```

-    function totalLiquidityTokenSupply() public view returns (uint256)
    {
+    function totalLiquidityTokenSupply() external view returns (
   uint256) {
```

[I-10] TITLE (Root Cause + Impact) In `TSwapPool::getPriceOfOneWethInPoolTokens` constants should be used instead of magic numbers

```
uint256 public constant ONE_UNIT = 1e18;
-         1e18,
+         ONE_UNIT
        i_wethToken.balanceOf(address(this)),
```

[I-11] TITLE (Root Cause + Impact) In `TSwapPool::getPriceOfOnePoolTokenInWeth` constants should be used instead of magic numbers

```
uint256 public constant ONE_UNIT = 1e18;
-         1e18,
+         ONE_UNIT
        i_poolToken.balanceOf(address(this)),
```