TSwap Audit Report

Version 1.0

Max

December 10, 2025

Protocol Audit Report

Max

12 10, 2025

Prepared by: Max Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The YOUR_NAME_HERE 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

Likelihood / Impact	High	Medium	Low
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

Scope

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	5
Medium	3

Severity	Number of issues found
Low	3
Info	7
Total	18

Findings

HIGH

[H-1] Send ETH to msg.sender in _swap, which breaks the invariant `x * y = k`

description in `_swap` function, the protocal send 1 outputToken to msg.sender every 10 transactions, which breaks the invariant of `x * y = k`.

Impact the invariant is broken!

Prove Of Concepts Add following code into codebase:

PoC

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;

import {Test, console} from "forge-std/Test.sol";
import {TSwapPool} from "../../src/TSwapPool.sol";
import {ERC20Mock} from "../mock/ERC20Mock.sol";

contract Handler is Test {
    TSwapPool pool;
    ERC20Mock poolToken;
    ERC20Mock weth;

    //Ghost Variable
    // x => weth
    int256 public startingX;
    int256 public expectedDeltaX;
    int256 public actualDeltaX;

    // y => poolToken
    int256 public startingY;
    int256 public expectedDeltaY;
    int256 public actualDeltaY;

    address liquidityProvider = makeAddr("liquidityProvider");
    address user = makeAddr("user");

    constructor(TSwapPool _pool) {
        pool = _pool;
        poolToken = ERC20Mock(pool.getPoolToken());
        weth = ERC20Mock(pool.getWeth());
    }

    function swapInputPoolTokenOutputWeth(uint256 outputWethAmount) public {
        if (
            weth.balanceOf(address(pool)) <= pool.getMinimumWethDepositAmount()
        ) {
            return;
        }
        outputWethAmount = bound(
            outputWethAmount,
            pool.getMinimumWethDepositAmount(),
            weth.balanceOf(address(pool))
        );
        if (outputWethAmount == weth.balanceOf(address(pool))) {
            return;
        }
        uint256 inputPoolTokenAmount = pool.getInputAmountBasedOnOutput(
            outputWethAmount,
            poolToken.balanceOf(address(pool)),
            weth.balanceOf(address(pool))
        );
        if (inputPoolTokenAmount >= type(uint64).max) {
            return;
        }
        _updateStartingDelta(
            -1 * int256(outputWethAmount),
            int256(inputPoolTokenAmount)
        );

        if (pool.balanceOf(user) < inputPoolTokenAmount) {
            poolToken.mint(
                user,
                uint256(inputPoolTokenAmount - poolToken.balanceOf(user) + 1)
            );
        }
        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);

        pool.swapExactOutput({
            inputToken: poolToken,
            outputToken: weth,
            outputAmount: outputWethAmount,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();
        _updateEndingDelta();
    }

    function deposit(uint256 wethAmountToDeposit) public {
        // setup
        wethAmountToDeposit = bound(
            wethAmountToDeposit,
            pool.getMinimumWethDepositAmount(),
            type(uint64).max
        );
        uint256 poolTokenToDeposit = pool.
            getPoolTokensToDepositBasedOnWeth(
            wethAmountToDeposit
        );
        _updateStartingDelta(
            int256(wethAmountToDeposit),
            int256(poolTokenToDeposit)
        );

        vm.startPrank(liquidityProvider);
        //buy token
        weth.mint(liquidityProvider, wethAmountToDeposit);
        poolToken.mint(liquidityProvider, poolTokenToDeposit);
        // deposit
        weth.approve(address(pool), wethAmountToDeposit);
        poolToken.approve(address(pool), poolTokenToDeposit);
        pool.deposit(
            wethAmountToDeposit,
            0,
            poolTokenToDeposit,
            uint64(block.timestamp)
        );
        vm.stopPrank();
        _updateEndingDelta();
    }

    function _updateStartingDelta(
        int256 wethAmount,
        int256 poolTokenAmount
    ) internal {
        // start
        startingX = int256(weth.balanceOf(address(pool)));
        startingY = int256(poolToken.balanceOf(address(pool)));

        // dalta
        expectedDeltaX = wethAmount;
```

```
            expectedDeltaY = poolTokenAmount;
        }

        function _updateEndingDelta() internal {
            uint256 endingX = uint256(weth.balanceOf(address(pool)));
            uint256 endingY = uint256(poolToken.balanceOf(address(pool)));

            int256 actualDeltalWeth = int256(endingX) - int256(startingX);
            int256 actualDeltaPoolToken = int256(endingY) - int256(
                startingY);

            actualDeltaX = actualDeltalWeth;
            actualDeltaY = actualDeltaPoolToken;
        }
    }
```

Recommended mitigation

1. delect related code in `_swap` function

```
- uint256 private constant SWAP_COUNT_MAX = 10;

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

[H-2] The `PoolFactory::createPool` Lack Of Role Check when create a pool, which will allow any wired ERC20 token to participate in the protocal and break the invariant.

Description any one can create a pool without permission, which allow some malicious ERC20 to take part in the protocal. Eg. Rebase Token can brake the invariant.

Prove Of Code

function `createPool` in `PoolFactory.sol`, as followed:

Found

```
function createPool(address tokenAddress) external returns (address) {
    ...code...
}
```

Recommended Mitigation

1. add some role check modifier

```
+ modifier onlyGov(address _addr) {
```

```
+     ...
+ }

+ function createPool(address tokenAddress)
+   external
+   onlyGov(msg.sender)
+   returns (address) {...}
```

[H-3] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost fees

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

Impact: Protocol takes more fees than expected from users.

Recommend 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-4] `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 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 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 protocol functionality.

`Proof of Concept`:

`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-5] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens

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 a maxInputAmount.

Impact: If market conditions change before the transaction processes, the user could get a much worse swap.

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,
+       uint256 maxInputAmount,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(outputAmount,
            inputReserves, outputReserves);
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
```

MEDIUM

[M-1] Lack Of Zero Check

Description have to chech zero when init the pool

Impact the poolFactory will fail.

Prove Of Concepts

code

```
constructor(address wethToken) {
        i_wethToken = wethToken;
    }
```

[M-2] The `PoolFactory::createPool` No Check for the poolToken's name and Symbool,causing the original pool invalid to visit

Description: In `PoolFactory::createPool` didn't check the new token's name and symbol. If it have an old tokenPool use the same name and symbol, the new one will coverage it.

Impact: It will cause the old pool couldn't be found through `getPool` or `getToken`.

Recommended Mitigation devopts should add some check before update the `s_pools` and `s_tokens`

[M-3] The `TSwapPool::deposit` is missing the deadline check, causing transactions to complete even after the deadline

Description The `deposit` function accepts a deadline parameter, which according to the documentation is "The deadline for the transaction to be completed by" . However, the parameter is never used. As a consequence, operations that add liqulidity to the pool may be excuted at unexpected times, in market conditions where the deposit rate is unfavorable //MEV

Impacts: Transactions could be sent when market conditions are unfavorable to deposit.

Recommended Mitigation Consider making the following change to the function:

```
function deposit(
    uint256 wethToDeposit,
    uint256 minimumLiquidityTokensToMint,
    uint256 maximumPoolTokensToDeposit,
    //@audit-HIGH: no check for the deadline
    uint64 deadline
)
    external
    //@audit-gas duplicated, which will increase the gas fee || already
        reported in [G-1]
    revertIfZero(wethToDeposit)
+   revertIfDeadlinePassed(deadline)
    returns (uint256 liquidityTokensToMint)
{

}
```

LOW

[L-1] Public Function Not Used Internally

If a function is marked public but is not used internally, consider marking it as `external`.

1 Found Instances

- Found in src/TSwapPool.sol Line: 304

```
	function swapExactInput()
```

[L-2] Input parameter transmission error in `emit LiquidityAdded`

[L-3] 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 caller. However, while it declares the named return value output it is never assigned a value, nor uses an explicit return statement. **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);
+        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);
}
}
```

GAS

[G-1] Dupplicated check, in resoult of little more gas fee

double chech to the `wethToDeposit`

Found

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
```

```
    )
        external
        //@audit-gas duplicated, which will increase the gas fee
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
                MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
        }
        ...
    }
```

Recommended Mitigation

1. delete the `revertIfZero`

INFO

[I-1]Unused Error

Consider using or removing the unused error.

1 Found Instances

- Found in src/PoolFactory.sol Line: 22

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

[I-2] Large Numeric Literal

Large literal values multiples of 10000 can be replaced with scientific notation.Use `e` notation, for example: `1e18`, instead of its full numeric value.

3 Found Instances

- Found in src/TSwapPool.sol Line: 46

```
    uint256 private constant MINIMUM_WETH_LIQUIDITY = 1
       _000_000_000;
```

- Found in src/TSwapPool.sol Line: 300

```
                ((inputReserves * outputAmount) * 10000) /
```

- Found in src/TSwapPool.sol Line: 409

```
                outputToken.safeTransfer(msg.sender, 1
                    _000_000_000_000_000_000);
```

[I-3] Literal Instead of Constant

Define and use `constant` variables instead of using literals. If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

4 Found Instances

- Found in src/TSwapPool.sol Line: 282

```
            uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 301

```
                ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 461

```
                    1e18,
```

- Found in src/TSwapPool.sol Line: 470

```
                    1e18,
```

[I-4] the MINIMUM_WETH_LIQUIDITY is constant, don't put it into emit

devopts can not use `MINIMUM_WETH_LIQUIDITY` because it is constant

```
revert TSwapPool__WethDepositAmountTooLow(
    MINIMUM_WETH_LIQUIDITY,
    wethToDeposit
);
```

[I-5] Don't Need This Param

the `poolTokenReserves` is never used and it is useless

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

[I-6] Follow The CEI

the `_addLiquidityMintAndTransfer` includes external calls, it would be batter if this is before the `_addLiquidityMintAndTransfer` to follow CEI

[I-7] Add some nespects in `swapExactInput`

add some nespect! to explaim the use of function `swapExactInput`

```
//nespects here
function swapExactInput(
    IERC20 inputToken,
    uint256 inputAmount,
    IERC20 outputToken,
    uint256 minOutputAmount,
    uint64 deadline
)
    public
    revertIfZero(inputAmount)
    revertIfDeadlinePassed(deadline)
    returns (
        //@audit-LOW: never use
        uint256 output
    )
{
}
```