Protocol Audit Report

Version 1.0

gf042

December 30, 2024

Protocol Audit Report

gf042

December 30, 2024

Prepared by: gf042

Table of Contents

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

Risk Classification

		Impact		
		High	Medium	Low
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

Scope

- Commit Hash: 1ec3c30253423eb4199827f59cf564cc575b46db
- In Scope:

```
./src/
#-- PoolFactory.sol
#-- TSwapPool.sol
```

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum

Executive Summary

Issues found

- 4 High severity findings
- 1 Medium severity finding
- 8 Informational findings
- Total: 13 findings

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput()` causes protocol to take too many tokens from users

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 byt 10_000 instead of 1_000.

Impact: Protocol takes more fees than expected.

Proof of Concept:
In `getInputAmountBasedOnOutput`, the fee calculation uses incorrect multiplier:

```
1 return ((inputReserves outputAmount) 10000) / ((outputReserves -
    outputAmount) 997);
```

Recommendation:

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

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput()` causes users to potentially receive much less tokens

Description: The `swapExactOutput()` function does not include any slippage protection. This means that users may receive significantly less tokens than expected, potentially leading to finan-

cial losses. This function is similar to what is done in `TSwapPool::swapExactInput()`, where the function specifies a `minOutputAmount` parameter, the `swapExactOutput` should specify a `maxInputAmount` parameter.

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

Proof of Concept: 1. The price of WETH is 1000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. minOutputAmount = 1 WETH 4. deadline = ... 3. The function does not offer a maxInputAmount 4. As the transaction is pending in the mempool, the market changes and the price moves huge: 1 WETH is now 10000 USDC. 10x more than the user expected. 5. The transaction completes, but the user sent to the protocol 10000 USDC for 1 WETH, instead of 1000 USDC.

Recommendation: We should include a `maxInputAmount` parameter in the `swapExactOutput()` function. So the user only has to spend up to a specific amount, and can predict how much they will spend on the protocol.

```
1
2  function swapExactOutput(
3      IERC20 inputToken,
4  +   uint256 maxInputAmount,
5  .
6  .
7  .
8      inputAmount = getInputAmountBasedOnOutput(outputAmount,
           inputReserves, outputReserves);
9  +   if(inputAmount > maxInputAmount) {
10 +       revert();
11 +   }
12     _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] `TSwapPool::sellPoolTokens()` mismatches input and output tokens causing users to receive 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 ammount.

This is due to the fact that the `swapExactOutput` function is called whereas the `swapExactInput` function is expected.

Impact: Users will swap the wrong amount of tokens, which is a severe disruption of the protocol functionality.

Proof of Concept:

Recommendation: Consider changing the function to use `swapExactInput()` instead of `swapExactOutput()`.

```
  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,
    minWethToReceive, uint64(block.timestamp));
    }
```

Additionally we need to add a deadline to the functiuon, as there is currently no deadline.

[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 invarianty of `x * y = k` where `x` is the balance of pool tokens and `y` is the amount of WETH. This means, that whenever tha 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 mailiciouslt drain the protocol of funds by doing a lot of swaps and collecting the extra tokens.

Proof of Concept: 1. A user swaps 10 times, and collects the extra incentive of `1_000_000_000_000_000_000` . 2. That user continues to do swaps, until all the protocol funds are drained.

Proof Of Code

```
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;
```

```
8       poolToken.mint(user, 100e18);
9
10      vm.startPrank(user);
11
12      poolToken.approve(address(pool), type(uint256).max);
13      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
14      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
15      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
16      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
17      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
18      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
19      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
20      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
21      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
22
23      int256 startingY = int256(weth.balanceOf(address(pool)));
24      int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25      pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
26
27      vm.stopPrank();
28
29      uint256 endingY = weth.balanceOf(address(pool));
30
31      int256 actualDeltaY = int256(endingY) - int256(startingY);
32
33      assertEq(actualDeltaY, expectedDeltaY);
34  }
```

Recommendation: Remove the extra incentive in the `_swap()` function.

Medium

[M-1] `deadline` not being used in `Tswap::deposit()` function causing transaction 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, this parameter is never

used. As a consequence, operations that add liquidity to the pool might be executed at unexpected times, in market conditions that are not optimal for the user.

Impact: Transactions could be sent when market conditions are not optimal for the user even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is not used in the `deposit()` function.

Recommendation: Consider making the following changes:

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

Informationals

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

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

[I-2] Lacking zero address check in constructor of `PoolFactory` and `TSwapPool`

```
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    )
        ERC20(liquidityTokenName, liquidityTokenSymbol)
    {
+       require(poolToken != address(0), "TSwapPool: Pool token cannot be the zero address");
+       require(wethToken != address(0), "TSwapPool: WETH token cannot be the zero address");
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

```
1      constructor(address wethToken) {
2  +       require(wethToken != address(0), "PoolFactory: WETH token
       cannot be the zero address");
3          i_wethToken = wethToken;
4      }
```

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

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

[I-4] `TSwapPool::Swap` should be indexed

```
1  -   event Swap(address indexed swapper, IERC20 tokenIn, uint256
       amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
2  +   event Swap(address indexed swapper, IERC20 indexed tokenIn, uint256
       amountTokenIn, IERC20 indexed tokenOut, uint256 amountTokenOut);
```

[I-5] In `TSwapPool::deposit()` `poolTokenReserves` can be removed

Description: The `poolTokenReserves` variable is not used in the `deposit()` function. So it can be removed because it uses some gas.

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

[I-6] In `TSwapPool::_addLiquidityMintAndTransfer()` emit `LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);` is backwards

Description: The `LiquidityAdded` event is emitted with the parameters in the reverse order. `event LiquidityAdded(address indexed liquidityProvider, uint256 wethDeposited, uint256 poolTokensDeposited);`

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

[I-7] In `TSwapPool::getOutputAmountBasedOnInput()` there are magic numbers

Description: In `TSwapPool::getOutputAmountBasedOnInput()` there are magic numbers.

```
uint256 inputAmountMinusFee = inputAmount * 997;
uint256 numerator = inputAmountMinusFee * outputReserves;
uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
```

Recommendation: Add constants for the magic numbers.

```
+ uint256 constant FEE_DENOMINATOR = 1000;
+ uint256 constant FEE_NUMERATOR = 997;
```

[I-8] 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 explict return statement.

Impact: The return value will always be 0, giving incorrect information to the caller.

Recommendation:

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

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