T-Swap Protocol Audit Report

Version 1.0

GuireWire

July 14, 2024

Protocol Audit Report

GuireWire

July 14, 2024

Prepared by: GuireWire Lead Auditors: - GuireWire

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

Disclaimer

The GuireWire 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
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

Commit Hash:

```
1ec3c30253423eb4199827f59cf564cc575b46db
```

Scope

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

Roles

Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made. Users: Users who want to swap tokens.

Executive Summary

Audit deemed to be successful as we identified 2 high severity bugs and 1 informational bug.

We spent 4 hours on the audit using Foundry.

Issues found

Severity	Number of Issues Found
High	5
Medium	1
Low	2
Informational	6
Total	14

Note: The total number of issues found is 14, but there were a number of additional informational findings that were caught but weren't addressed in this report since we specified these with the Protocol team during a previous audit.

Findings

High

[H-1] `TSwapPool::deposit` is missing 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, this parameter is never used. As a consequence, operationrs that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact:

Transactions 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, // LP tokens -> if empty,
            we can pick 100% (100% == 17 tokens)
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocll 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.

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);
    }
```

Proof Of Concept:

As a result, users swapping tokens via the `swapExactOutput` function will pay far more tokens than expected for their trades. This becomes particularly risky for users that provide infinite allowance to the `TSwapPool` contract. Moreover, note that the issue is worsened by the fact that the `swapExactOutput` function does not allow users to specify a maximum of input tokens, as is described in another issue in this report.

It's worth noting that the tokens paid by users are not lost, but rather can be swiftly taken by liquidity providers. Therefore, this contract could be used to trick users, have them swap their funds at unfavorable rates and finally rug pull all liquidity from the pool.

To test this, include the following code in the `TSwapPool.t.sol` file:

```
function testFlawedSwapExactOutput() public {
    uint256 initialLiquidity = 100e18;
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), initialLiquidity);
    poolToken.approve(address(pool), initialLiquidity);

    pool.deposit({
        wethToDeposit: initialLiquidity,
        minimumLiquidityTokensToMint: 0,
        maximumPoolTokensToDeposit: initialLiquidity,
        deadline: uint64(block.timestamp)
    });
    vm.stopPrank();

    // User has 11 pool tokens
    address someUser = makeAddr("someUser");
    uint256 userInitialPoolTokenBalance = 11e18;
    poolToken.mint(someUser, userInitialPoolTokenBalance);
    vm.startPrank(someUser);

    // Users buys 1 WETH from the pool, paying with pool tokens
    poolToken.approve(address(pool), type(uint256).max);
    pool.swapExactOutput(
        poolToken,
        weth,
        1 ether,
        uint64(block.timestamp)
    );

    // Initial liquidity was 1:1, so user should have paid ~1 pool
       token
    // However, it spent much more than that. The user started with 11
       tokens, and now only has less than 1.
    assertLt(poolToken.balanceOf(someUser), 1 ether);
    vm.stopPrank();

    // The liquidity provider can rug all funds from the pool now,
```

```
36      // including those deposited by user.
37      vm.startPrank(liquidityProvider);
38      pool.withdraw(
39          pool.balanceOf(liquidityProvider),
40          1, // minWethToWithdraw
41          1, // minPoolTokensToWithdraw
42          uint64(block.timestamp)
43      );
44
45      assertEq(weth.balanceOf(address(pool)), 0);
46      assertEq(poolToken.balanceOf(address(pool)), 0);
47  }
```

[H-3] 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 transaciton 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
    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 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);
```

[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 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-5] 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.

```
1          swap_count++;
2          if (swap_count >= SWAP_COUNT_MAX) {
3              swap_count = 0;
4              outputToken.safeTransfer(msg.sender, 1
                   _000_000_000_000_000_000);
5          }
```

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 untill all the protocol funds are drained

Proof Of Code

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

```
1      function testInvariantBroken() public {
2          vm.startPrank(liquidityProvider);
3          weth.approve(address(pool), 100e18);
4          poolToken.approve(address(pool), 100e18);
5          pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
6          vm.stopPrank();
```

```
7
8           uint256 outputWeth = 1e17;
9
10          vm.startPrank(user);
11          poolToken.approve(address(pool), type(uint256).max);
12          poolToken.mint(user, 100e18);
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
26          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31          assertEq(actualDeltaY, expectedDeltaY);
32      }
```

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.

```
1 -         swap_count++;
2 -         // Fee-on-transfer
3 -         if (swap_count >= SWAP_COUNT_MAX) {
4 -             swap_count = 0;
5 -             outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
6 -         }
```

Medium

[M-1] Rebase, fee-on-transfer, ERC777, and centralized ERC20s can break core invariant

Description:

The core invariant of the protocol is:

`x * y = k`. In practice though, the protocol takes fees and actually increases k. So we need to make sure `x * y = k` before fees are applied.

Note: This is linked to the protocol invariant break on the function `_swap`. Will write this in the future

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order

Description:

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

Impact:

Event emission is incorrect, leading to off-chain functions potentially malfunctioning.

Recommended Mitigation:

```
1 - emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
2 + 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 caller. However, while it declares the named return value `ouput` 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.

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);
    }
```

Informational

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

```
- error 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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

```
1      event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 43

```
1      event LiquidityAdded(address indexed liquidityProvider,
          uint256 wethDeposited, uint256 poolTokensDeposited);
```

- Found in src/TSwapPool.sol Line: 44

```
1      event LiquidityRemoved(address indexed liquidityProvider,
          uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
```

- Found in src/TSwapPool.sol Line: 45

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

[I-5] Poor test coverage

```
1  Running tests...
2  | File                | % Lines         | % Statements    | % Branches
         | % Funcs       |
3  | ------------------- | --------------- | --------------- |
      ------------- | ------------- |
4  | src/PoolFactory.sol | 100.00% (11/11) | 100.00% (16/16) | 100.00%
     (2/2) | 100.00% (3/3) |
5  | src/TSwapPool.sol   | 54.84% (34/62)  | 59.14% (55/93)  | 33.33%
     (6/18) | 37.50% (6/16) |
```

Recommended Mitigation:

Aim to get test coverage up to over 90% for all files.

[I-6] Provide better error for `TSwapPool::getInputAmountBasedOnOutput`

When `outputReserves` & `outputAmount` are the same, `TSwapPool::getInputAmountBasedOnOutput` will revert with arithmetic divide by zero. This is not very helpful for users. Consider adding a custom error message.

“‘diff + error TSwapPool__CannotRemoveEntireBalance(); . . . + if (outputReserves, outputAmount) {revert TSwapPool__CannotRemoveEntireBalance();} return (inputReserves * outputAmount) / ((output-Reserves - outputAmount));