T-swap Report

Version 1.0

September 14, 2024

T-Swap Audit Report

Jaxon Chen

09/14/2024

Lead Auditors: Jaxon Chen

Table of Contents

    * [H-5] The invariant `x * y = k` breaks after user did a certain amount of swaps calling `TSwapPool::_swap`, which gives user extra tokens
  - Medium
    * [M-1] Non-standard ERC20 tokens (rebase, fee-on-transfer, ERC777) break protocol invariant
  - Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order, emitting incorrect information
    * [L-2] The value returned by `TSwapPool::swapExactInput` is never used and incorrect
  - Informationals
    * [I-1] `PoolFactory__PoolDoesNotExist` is not used and should be removed.
    * [I-2] **`public`** functions not used internally could be marked `external`
    * [I-3] `PoolFactory::constructor` lacks zero address checks
    * [I-4] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
    * [I-5] Define and use `constant` variables instead of using literals in `TSwapPool`
    * [I-6] Event is missing `indexed` fields
    * [I-7] Large literal values multiples of 10000 can be replaced with scientific notation
    * [I-7] It is recommended to follow CEI pattern
    * [I-8] Function `TSwapPool::swapExactInput` missing natspec

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

Author 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: e643a8d4c2c802490976b538dd009b351b1c8dda
- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
  - Any ERC20 token

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.

Issues found

Severity	Number of issues found
High	5
Medium	1
Low	2
Info	8
Total	16

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 paramter, which according to the documentation is the deadline for the transaction to be completed by. However, this parameter is never used. Operations that add liquidity to the pool might be executed at unexpected times.

Impact: Transactions could be sent when market conditions are unfavorable to deposit. Users may consider that the transaction is failed but it's actually processing. **Proof of Concept:**

1. The deposit function signature includes the deadline parameter.
2. However, the `deadline` parameter is never used within the function body. There's no check against `block.timestamp` to ensure the transaction is executed before the deadline.
3. This contrasts with other functions in the contract, such as `withdraw`, which correctly implements the deadline check:

```
    function withdraw(
        uint256 liquidityTokensToBurn,
        uint256 minWethToWithdraw,
        uint256 minPoolTokensToWithdraw,
        uint64 deadline
    )
        external
        revertIfDeadlinePassed(deadline)
        ...
```

Recommended Mitigation:

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

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` function leads to severe mispricing and potential loss of funds

Description: In the function `TSwapPool::getInputAmountBasedOnOutput`, there is a critical error in the fee calculation. The function uses 10000 as a multiplier instead of the correct value of 1000. This error results in a significant overestimation of the required input amount for a given output, leading to severe mispricing and potential exploitation.

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

Impact:

- Users will be charged approximately 10 times more input tokens than they should be for a given output amount.
- This overcharging will likely lead to a rapid drain of input tokens from users, potentially emptying their accounts unexpectedly.
- The pool's economics will be severely disrupted, as the ratio of tokens in the pool will become imbalanced.

- Attackers could potentially exploit this mispricing for significant profit at the expense of regular users and the protocol.

Proof of Concept:

1. Assume inputReserves = 1000 ETH, outputReserves = 1000 USDC, and a user wants to buy 100 USDC.
2. With the correct calculation (1000 multiplier), the input amount would be approximately 111.58 ETH.
3. With the incorrect calculation (10000 multiplier), the input amount would be approximately 1115.8 ETH.

This shows that users are being charged about 10 times more than they should be.

Recommended Mitigation:

Replace the `10000` multiplier with `1000` in the `getInputAmountBasedOnOutput` function.

[H-3] Function `TSwapPool::swapExactOutput` lacks of slippage protection causing users to potentially receive fewer tokens

Description: The function `swapExactOutput` does not have any slippage protection. This function acts similar roles as `TSwapPool::swapExactInput`, where the function specifies a `minOutputAmount`. The `swapExactOutput` should also specifies a `maxInputAmount` **Impact:** If market conditions drastically change before the transaction processes, the user could get a very worse swap. **Proof of Concept:**

1. Assume the price of 1 WETH is 2,000 USDC
2. User calls `swapExactOutput` for 1 WETH
    1. inputToken = USDC
    2. outputToken = WETH
    3. outputAmount = 1
3. The function does not offer a maxInput amount
4. As the transaction is pending in the mempool, the market changes. The price becomes 1 WETH = 10,000 USDC.
5. The transaction completes but the user sent 10,000 USDC instead of 2,000 USDC **Recommended Mitigation:** Add a `maxInputAmount` and revert if the transaction does not meet the value.

```
function swapExactOutput(
    IERC20 inputToken,
    IERC20 outputToken,
```

```
    uint256 outputAmount,
+   uint256 maxInputAmount,
    uint64 deadline
)
.
.
.
+   if (inputAmount > maxInputAmount) {
+       revert;
+   }

    _swap(inputToken, inputAmount, outputToken, outputAmount);
}
```

[H-4] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive wrong amounts of tokens

Description: The `sellPoolTokens` function is intended to allow users to easily sell pool tokens. Users indicate how many tokens they want to sell in the `poolTokenAmount` parameter. The `swapExactOutput` is called, whereas the `swapExactInput` is the correct function that should be called. This results a miscalculation. **Impact:** Users will swap the wrong amount of tokens, which is a severe break of protocol functionaility. **Proof of Concept:**

- Assume the pool has 1000 WETH and 2000 pool tokens
- A user wants to sell 100 pool tokens -The correct calculation would be: CopywethToReceive = getOutputAmountBasedOnInput(100, 2000, 1000) = 47.6 WETH
- But with the current implementation: CopypoolTokensToSell = getInputAmountBasedOnOutput(100, 2000, 1000) = 250 pool tokens

The function will try to sell 250 pool tokens instead of 100, and the user will receive 100 WETH instead of 47.6 WETH **Recommended Mitigation:** Replace the `swapExactOutput` call with `swapExactInput` in the `sellPoolTokens`

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

[H-5] The invariant `x * y = k` breaks after user did a certain amount of swaps calling `TSwapPool::_swap`, which gives user extra tokens

Description: The protocol defines the invariant `x * y = k` where:

- x := the balance of the pool token
- y := the balance of WETH
- k := an arbitrary constant

The `k` should always stay same no matter `x` and `y` changes. However, the bounty mechanism in `_swap` function breaks the rule by sending user extra tokens after `10` swaps.

Impact: Amalicious user could drain the protocol of funds by doing a lot of swaps and collecting extra incentive tokens.

Proof of Concept:

- A user swaps 10 times, and collects the extra tokens.
- Repeatedly doing this untill all the protocol funds are drained.

Proof of Code

Place below code into test file.

```
1
2     function testInvariantBroken() public {
3         vm.startPrank(liquidityProvider);
4         weth.approve(address(pool), 100e18);
5         poolToken.approve(address(pool), 100e18);
6         pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7         vm.stopPrank();
8
9         uint256 outputWeth = 1e17;
10
11        vm.startPrank(user);
12        poolToken.approve(address(pool), type(uint256).max);
13        poolToken.mint(user, 100e18);
14        for (uint256 i; i < 10; i++){
15            pool.swapExactOutput(poolToken, weth, outputWeth, uint64(
                  block.timestamp));
16        }
17
18        int256 startingY = int256(weth.balanceOf(address(pool)));
19        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
20
21        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
22        vm.stopPrank();
23
```

```
24          uint256 endingY = weth.balanceOf(address(pool));
25          int256 actualDeltaY = int256(endingY) - int256(startingY);
26          assertEq(actualDeltaY, expectedDeltaY);
27      }
```

Recommended Mitigation: Remove the extra incentive mechanism.

```
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] Non-standard ERC20 tokens (rebase, fee-on-transfer, ERC777) break protocol invariant

Description: The TSwapPool protocol assumes standard `ERC20` token behavior for both the pool token and WETH. However, it does not account for non-standard token implementations such as rebase tokens, fee-on-transfer tokens, or ERC777 tokens. These token types can alter balances in unexpected ways, breaking the core invariant of the protocol: `x * y = k`. **Impact:** The use of non-standard tokens can lead to:

- Incorrect pricing calculations
- Unfair distribution of liquidity tokens
- Potential loss of funds for liquidity providers
- Exploitation opportunities for malicious actors **Proof of Concept:**

1. Rebase Tokens:
    - A rebase token could increase its supply, artificially inflating the pool's balance without any actual deposits.
    - Example: If the pool contains 1000 rebase tokens and 1000 WETH, and a rebase occurs doubling the supply, the pool now has 2000 rebase tokens and 1000 WETH, breaking the x * y = k invariant.
2. Fee-on-Transfer Tokens:
    - These tokens deduct a fee on each transfer, meaning the received amount is less than the sent amount.

- Example: If a user deposits 1000 tokens with a 1% transfer fee, the pool only receives 990 tokens, but the calculations assume 1000 tokens were added.

3. ERC777 Tokens:

```
- These tokens have hooks that can execute arbitrary code during transfers.
- A malicious token could manipulate the pool's state or reenter functions during a swap or liquidity provision, potentially draining the pool.
```

Recommended Mitigation:

1. disallow non-standard ERC20 tokens: Implement a whitelist of approved tokens that have been verified to be standard ERC20 implementations. In the constructor and any functions that interact with tokens, check if the token is on the whitelist.
2. Implement balance checks: Before and after every token transfer, check the actual balance change of the contract. Use these real balance changes in calculations instead of the expected transfer amounts.
3. Use safe transfer libraries: Ensure all token transfers use safe transfer methods that check for success and revert on failure.
4. Implement reentrancy guards: Use OpenZeppelin's ReentrancyGuard or similar to prevent potential reentrancy attacks from ERC777 tokens

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order, emitting incorrect information

Description: When the event `LiquidityAdded` is emitted in the function `_addLiquidityMintAndTransfer`, it logs values in an incorrect order. The `poolTokensToDeposit` should be in the thrid postion and the `wethToDeposit` should be in the second. **Impact:** Leading misunderstanding or off-chain applications malfunctioning

Recommended Mitigation:

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

[L-2] The value returned by `TSwapPool::swapExactInput` is never used and incorrect

Description: The `swapExactInput` function is expected to return the actual amount of tokens bought by the user. However, it declares the name retun value `output` but never assigned a value.

Impact: The `output` will always be 0, giving incorrect information to the user.

Recommended Mitigation:

```
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));

-   uint256 outputAmount = getOutputAmountBasedOnInput(
+   output = getOutputAmountBasedOnInput(
        inputAmount,
        inputReserves,
        outputReserves
    );
-   if (outputAmount < minOutputAmount) {
+   if (output < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount,
               minOutputAmount);
        }

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

Informationals

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

it is recommended that the definition be removed when custom error is unused

1 Found Instances

- Found in src/PoolFactory.sol Line: 22

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

[I-2] `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

1 Found Instances

- Found in src/TSwapPool.sol Line: 298

```
    function swapExactInput()
```

[I-3] `PoolFactory::constructor` lacks zero address checks

```
constructor(address wethToken) {
+   if(wethToken == address(0)){
+       revert();
+   }
    i_wethToken = wethToken;
    }
```

[I-4] `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-5] Define and use `constant` variables instead of using literals in `TSwapPool`

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: 276

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

- Found in src/TSwapPool.sol Line: 295

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

- Found in src/TSwapPool.sol Line: 454

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
                1e18,
```

[I-6] 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: 52

```
1       event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1       event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1       event Swap(
```

[I-7] 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: 45

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

- Found in src/TSwapPool.sol Line: 294

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

- Found in src/TSwapPool.sol Line: 402

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

[I-7] It is recommended to follow CEI pattern

```
else {
            // This will be the "initial" funding of the protocol. We are starting from blank here!
            // We just have them send the tokens in, and we mint liquidity tokens based on the weth
-           _addLiquidityMintAndTransfer(
-               wethToDeposit,
-               maximumPoolTokensToDeposit,
-               wethToDeposit
-           );
            liquidityTokensToMint = wethToDeposit;
+           _addLiquidityMintAndTransfer(
+               wethToDeposit,
+               maximumPoolTokensToDeposit,
+               wethToDeposit
+           );
        }
```

[I-8] Function `TSwapPool::swapExactInput` missing natspec