APEX
SECURITY

TSwap Audit Report

Version 1.0

Austin Patkos

February 7, 2024

Prepared by: APex Lead Auditors: - Austin Patkos

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

Austin Patkos 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
	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

Audit Scope Details

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope:

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

- 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.

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many too many tokens form user, resulting in lost fees.

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should eposit given an amount of output tokens. However the function currently

miscalculates the resulting amount. When calculating the fee, it scales the amoutn 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)
      {
          // 91.3% fee???
          //@audit - high
          return
-             ((inputReserves * outputAmount) * 10000) /
+            ((inputReserves * outputAmount) * 1000) /
              ((outputReserves - outputAmount) * 997);
      }
```

[H-2] Lack of slippage protectyion 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 in `TSwapPool::swapExactInput`, where the function specifies a `minOutputAmount`, the `swapExactOutput` funciton should specify a `maxInputAmount`.

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

Proof of Concept: 1. The price of WETH right now is 1000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. input = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = whatever 3. The function does not offer a maxInput amount 4. As the transactino is pending in the mempool, the market changes!! Andthe price moves HUGE -> 1 WETH is now 10,000 USDC. 10 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 toa specific amount, and can predict how much they will spend on the protocol.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,
.
.
```

```
6  .
7          inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
8  +       if(inputAmount > maxInputAmount) {
9  +    revert();
10 +}
11         _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] The `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 exhagne. User indicates how many pool tokens they're willing to sell in the `poolTokenAmount` parameter. However, the funciton currently miscalculates the swapped amount.

This is due to the fact the `swapExactOutput` function is called whereas the `swapEactInput` fuction is the one that should be called. Because users specifyt the exact amount of input tokens, not outpout.

Impact: Users qwill swap the wwrong 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 this would also require changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`);

```
1      function sellPoolTokens( uint256 poolTokenAmount
2  +   uint256 minWethToReceive
3      ) external returns (uint256 wethAmount) {
4  -        return swapExactOutput(i_poolToken,i_wethToken,
       poolTokenAmount,uint64(block.timestamp));
5  +        return swapExactOutput(i_poolToken, poolTokenAmount,
       minWethToReceive, uint64(block.timestamp));
6      }
```

Additionally it might be wise to add a deadline tothe function, as there is no deadline.

[H-4] In `TSwapPool::_swap` the extra tokens give to users afer every `swapCount` breaks the protocol invariant of `x * y = k`

Description: The protocol follows a sctrict 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 wenever 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 protcol funds will be drained.

The following block of code is responsiblefor 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 malicously drain drain the protocol of funds by doing a lot of swaps and collectin the extra incentive give out by the protocol.

More simply put, the protocols core invariant is borken.

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 until all the protocol funds are drained.

Proof of code

Place the following into `TSwapPool.t.sol`

```
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        int256 startingY = int256(poolToken.balanceOf(address(pool)));
11        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
12
13        vm.startPrank(user);
14        poolToken.approve(address(pool), type(uint256).max);
15        poolToken.mint(user, 100e18);
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          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
23          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
24          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
25          vm.stopPrank();
26
27          uint256 endingY = weth.balanceOf(address(pool));
28          int256 actualDetlaY = int256(endingY) - int256(startingY);
29
30          assertEq(actualDetlaY, expectedDeltaY);
31      }
```

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 tokensin the same way we do with fees.

```
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 -         }
```

Medium

[M-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 “@param deadline The deadline for the transaction to be completed by”. However, this parameter is never used. As a consequence, operations that add liquidity tothe pool might be executed at times, in market conditions where the deposte rate is unfavorable.

Impact: Transactions could be sent when market conditions are unfavorable to deposit, even adding 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,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+        revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[M-2]Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant.

Low

[L-1] `TSwapPool::Liqudiity` has parameters outof order causing event to emit incorrect information.

Description: When the `Liqudiity` event is emiited in `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 incorerct, leading to off-chain functions potentially malfunctioning.

Recommended Mitigation:

```
-emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+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 `output` it is nefver assigned value, nor uses an explict return statement.

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

Proof of Concept:

Recommended Mitigation:

```
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
-       uint256 outputReserves = outputToken.balanceOf(address(this));
+       output = outputToken.balanceOf(address(this));

        uint256 outputAmount = 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);
    }
```

Gas

Informationals

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

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

[I-2] Constructor of `PoolFactory` is lacking zero address check.

```
    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]: Events in `TSwapPool` are 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.

Instances

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
    event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
    event Swap(
```