Prepared by: Eric Sullivan Lead Auditors: - Eric Sullivan

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.

Disclaimer

The Sullivan CORP 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
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L

	Impact			
	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: e643a8d4c2c802490976b538dd009b351b1c8dda In 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

A week long audit, using many tools like Slyther, Aderyn, building a test suite to break the core invariant, and finally doing a manual review. We found many issues detailed below:

Issues found

Severity	Number of Issues found
HIGH	4
MEDIUM	2
LOW	2
INFO	5
TOTAL	13

Findings

HIGH

[H-1] 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.

Proof of Concept:

Recommended Mitigation:

```
1
2   function getInputAmountBasedOnOutput(
3           uint256 outputAmount,
4           uint256 inputReserves,
5           uint256 outputReserves
6       )
7           public
8           pure
9           revertIfZero(outputAmount)
10          revertIfZero(outputReserves)
11          returns (uint256 inputAmount)
12      {
13  -           return ((inputReserves * outputAmount) * 10_000) /
        ((outputReserves - outputAmount) * 997);
14  +           return ((inputReserves * outputAmount) * 1_000) /
        ((outputReserves - outputAmount) * 997);
15      }
```

[H-2] 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: The price of 1 WETH right now is 1,000 USDC

User inputs a swapExactOutput looking for 1 WETH

inputToken = USDC

outputToken = WETH

outputAmount = 1

deadline = whatever

The function does not offer a maxInput amount

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

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.

```
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 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)

```
1
2 function sellPoolTokens(
3         uint256 poolTokenAmount,
4 +       uint256 minWethToReceive,
5         ) external returns (uint256 wethAmount) {
6 -        return swapExactOutput(i_poolToken, i_wethToken,
     poolTokenAmount, uint64(block.timestamp));
7 +        return swapExactInput(i_poolToken, poolTokenAmount
     , i_wethToken, minWethToReceive, uint64(block.timestamp)
     );
8      }
```

[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 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 until all the protocol funds are drained

Place the following into TSwapPool.t.sol.

Proof of code

Place the following into 'TSwapPool.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
11         vm.startPrank(user);
12         poolToken.approve(address(pool), type(uint256).max)
               ;
13         poolToken.mint(user, 100e18);
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         pool.swapExactOutput(poolToken, weth, outputWeth,
               uint64(block.timestamp));
23
24         int256 startingY = int256(weth.balanceOf(address(
               pool)));
25         int256 expectedDeltaY = int256(-1) * int256(
               outputWeth);
26
27         pool.swapExactOutput(poolToken, weth, outputWeth,
               uint64(block.timestamp));
28         vm.stopPrank();
29
30         uint256 endingY = weth.balanceOf(address(pool));
31         int256 actualDeltaY = int256(endingY) - int256(
```

```
            startingY);
32          assertEq(actualDeltaY, expectedDeltaY);
33      }
```

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
2 -         swap_count++;
3 -         // Fee-on-transfer
4 -         if (swap_count >= SWAP_COUNT_MAX) {
5 -             swap_count = 0;
6 -             outputToken.safeTransfer(msg.sender, 1
    _000_000_000_000_000_000);
7 -         }
```

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 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 where the deposit rate is unfavorable.

Impact: Transactions could be sent when market conditions are unfavorable, even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is unused. Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning. –> src/TSwapPool.sol:96:9: | 96 | uint64 deadline | ^^^^^^^^^^^^^^^^

Recommended Mitigation:

```
1
2 function deposit(
3         uint256 wethToDeposit,
4         uint256 minimumLiquidityTokensToMint,
5         uint256 maximumPoolTokensToDeposit,
6         uint64 deadline
7     )
8         external
9 +       revertIfDeadlinePassed(deadline)
10        revertIfZero(wethToDeposit)
11        returns (uint256 liquidityTokensToMint)
12    {...}
```

[M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariant

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

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. When it comes to auditing smart contracts, there are a lot of nitty-gritty details that one needs to pay attention to in order to prevent possible vulnerabilities.

Recommended Mitigation:

```
1
2 - emit LiquidityAdded(msg.sender, poolTokensToDeposit,
     wethToDeposit);
3 + 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 never assigned a value, nor uses an explicit return statement.

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

Proof of Concept:

Recommended Mitigation:

```
1
2 {
3     uint256 inputReserves = inputToken.balanceOf(address(
        this));
```

```
4     uint256 outputReserves = outputToken.balanceOf(address(
          this));
5
6 -         uint256 outputAmount = getOutputAmountBasedOnInput
      (inputAmount, inputReserves, outputReserves);
7 +         output = getOutputAmountBasedOnInput(inputAmount,
      inputReserves, outputReserves);
8
9 -         if (output < minOutputAmount) {
10 -            revert TSwapPool__OutputTooLow(outputAmount,
      minOutputAmount);
11 +         if (output < minOutputAmount) {
12 +            revert TSwapPool__OutputTooLow(outputAmount,
      minOutputAmount);
13      }
14
15 -         _swap(inputToken, inputAmount, outputToken,
      outputAmount);
16 +         _swap(inputToken, inputAmount, outputToken, output
      );
17  }
18  }
```

Informationals

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

```
1
2 - error PoolFactory__PoolDoesNotExist(address tokenAddress)
      ;
```

[I-2] PoolFactory::constructor Lacking zero address check

```
1
2 constructor(address wethToken) {
3 +    if(wethToken == address(0)){
4 +     revert();
5 +}
6      i_wethToken = wethToken;
7 }
```

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

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

```
3 + string memory liquidityTokenSymbol = string.concat("ts",
     IERC20(tokenAddress).symbol());
```

[I-4] TSwapPool::constructor Lacking zero address check - wethToken & poolToken

```
1
2 constructor(
3     address poolToken,
4     address wethToken,
5     string memory liquidityTokenName,
6     string memory liquidityTokenSymbol
7 )
8     ERC20(liquidityTokenName, liquidityTokenSymbol)
9 {
10 +    if(wethToken || poolToken == address(0)){
11 +        revert();
12 +    }
13      i_wethToken = IERC20(wethToken);
14      i_poolToken = IERC20(poolToken);
15 }
```

[I-5] TSwapPool events should be indexed

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

Bonus Aderyn findings

H-1: Reentrancy: State change after external call

Changing state after an external call can lead to re-entrancy attacks.Use the checks-effects-interactions pattern to avoid this issue.

2 Found Instances

- Found in src/PoolFactory.sol Line: 51

  State is changed at: `s_pools[tokenAddress] = address(tPool)`, `s_tokens[address(tPool)] = tokenAddress`

```
1          string memory liquidityTokenName = string.
              concat("T-Swap ", IERC20(tokenAddress).name
              ());
```

- Found in src/PoolFactory.sol Line: 52

  State is changed at: `s_pools[tokenAddress] = address(tPool)`, `s_tokens[address(tPool)] = tokenAddress`

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

Low Issues

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: 298

```
1      function swapExactInput(
```

L-2: 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: 276

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

- Found in src/TSwapPool.sol Line: 295

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

- Found in src/TSwapPool.sol Line: 454

```
1                  1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
1                  1e18,
```

L-3: PUSH0 Opcode

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

2 Found Instances

- Found in src/PoolFactory.sol Line: 15

```
1 pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
1 pragma solidity 0.8.20;
```

L-4: 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: 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);
```

L-5: Unused Error

Consider using or removing the unused error.

1 Found Instances

- Found in src/PoolFactory.sol Line: 22

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