Protocol Audit Report

Version 1.0

Ease Industries

July 15, 2024

Protocol Audit Report

Ease Industries

June 20, 2024

Prepared by: Ease Industries

Lead Auditors: - Eyan Ingles

Table of Contents

Protocol Summary

`T-Swap` is a protocol designed to facilitate liquidity pools for general users to swap between two or more different ERC-20 tokens. The protocol incentivizes liquidity providers by offering a percentage fee each time a user performs a swap. Additionally, `T-Swap` enables the creation of new pools, which can be used for swapping various tokens.

Disclaimer

Eyan Ingles 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

Scope

- 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 ## Actors / 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

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	7
total	14

Findings

High

[H-1] incorrect fee calculation on `TSwapPool::getInputAmountBasedOnOutput` causing protocol to take too many tokens from users, resulting in lost fees

Description: The `getInputAmountBasedOnOutput` function is intendedto calculate the amount of tokens a user should deposit given an amount of tokens of output tokens. The function is miscalculationg the resulting amount by scaling it at 10_000 instead of 1_000.

Impact: Protocol tkaes more fees than expected from users.

Proof of Concept: In the following proof of code provided below. 1. We start with the liquidity Provider put into the pool with an even amount of each token giving the pool a conversion of 1-10 (1 weth = 10 poolToken). 2. `someUser` comes along and mints 100 poolTokens and now wants to swap to get some weth tokens. 3. user approves swap of poolToken amount and then calls the `swapExactOutput` with the amount of weth wanting is 1. 4. We now have successfully done the swap and need to check the balances of before and after.

```
0 balance before trade. (weth)
100 balance before trade. (pooltoken)
1 balance after trade. (weth)
90 balance after trade. (pooltoken)
```

We now clearly see that the tokens taken out as to what was expected is 10X as much, protocol is taking too many tokens for fees.

```
    function testIncorrectFeeCalculation() public {

        // LP provides tokens into pool giving an even ratio.
        vm.startPrank(liquidityProvider);
```

```
5           weth.approve(address(pool), 100e18);
6           poolToken.approve(address(pool), 100e18);
7           pool.deposit(100 ether, 100 ether, 100 ether, uint64(block.
                timestamp));
8           vm.stopPrank();
9           // new user comes along and mints some poolTokens to do a swap
                to get some weth.
10          address someUser = makeAddr("someUser");
11          poolToken.mint(address(someUser), 100);
12          // shows us the balance before swap and after then minting of
                poolTokens.
13          console.log(weth.balanceOf(address(someUser)), "balance before
                trade. weth"); // balance = 0
14          console.log(poolToken.balanceOf(address(someUser)), "balance
                before trade. pooltoken"); // balance = 100
15          // someUser interacts with contract.
16          vm.startPrank(someUser);
17          poolToken.approve(address(pool), 10e18);
18          // someuser wants to swap 1 poolToken for 1 weth.
19          pool.swapExactOutput(poolToken, weth, 1, uint64(block.timestamp
                ));
20          vm.stopPrank();
21          // shows us how many
22          console.log(weth.balanceOf(address(someUser)), "balance after
                trade. weth"); // balance = 1
23          console.log(poolToken.balanceOf(address(someUser)), "balance
                after trade. pooltoken"); // balance = 90
24      }
```

PoC

Recommended Mitigation:

```
1 -         return ((inputReserves * outputAmount) * 10000) / ((
      outputReserves - outputAmount) * 997);
2 +         return ((inputReserves * outputAmount) * 1_000) / ((
      outputReserves - outputAmount) * 997);
```

[H-2] `TSwapPool::swapExactOutput` does not have any slippage controls, causing a user to swap using more tokens then expected.

Description: The `swapExactOutput` function does not limit the input tokens amount allowing for a major change in market conditions to swap using more tokens then expected, leaving the user to pay more for the same amount as to what was expected or calculated.

Impact: Users will spend more tokens to get the expected amount making swapping more costly in the event of changes to the market conditions.

Proof of Concept: 1. liquidityProvider provides into the pool for users to trade. 2. a user comes along and starts to calculate how much it will cost in pool tokens for some weth. User is expecting to trade 10 poolTokens to receieve 3 weth according to the calculator. 3. a whale comes in, mints poolTokens and swaps a large amount for weth, changing the market conditions for the user. 4. The user now swaps their poolTokens for the desired weth amount. 5. The user token balances have been updated, weth balance is 3 as expected but we had spent 114 tokens instead of the inital expected cost of 10 poolTokens.

Test is below with console.logs to see what the values are while the swaps are happening.

```
    function testForNoSlippage() public {
        // LP deposits into pool.
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 200e18);
        poolToken.approve(address(pool), 200e18);
        pool.deposit(200 ether, 200 ether, 200 ether, uint64(block.
            timestamp));
        vm.stopPrank();
        // new address mints tokens ready for deposit
        address user1 = makeAddr("user1");
        poolToken.mint(address(user1), 1e5);
        // check user1 balance
        console.log("user1 balance:", poolToken.balanceOf(user1));
        // swapExactOutput is the fault function
        // user1 starts the swap by getting an estimated swap of tokens
             and how many.
        vm.startPrank(user1);
        uint256 estimatedOutput = pool.getOutputAmountBasedOnInput(
            10, //input 10 tokens
            20, // allowing 20 tokens for reserve
            10
        );
        vm.stopPrank();
        //user1 has now got an estimated value of the swap
        // LP will now take put a large amount of tokens out of the
            pool affecting the value of exchange of each tokens
        poolToken.mint(liquidityProvider2, 300e18);
        weth.mint(liquidityProvider2, 300e18);

        console.log("pool balance of weth before whale:", weth.
            balanceOf(address(pool)));
        console.log("pool balance of poolToken before whale:",
            poolToken.balanceOf(address(pool)));
        vm.startPrank(liquidityProvider2);
        poolToken.approve(address(pool), 300e18);
        weth.approve(address(pool), 300e18);
        pool.swapExactInput(poolToken, 190 ether, weth, 5 ether, uint64
            (block.timestamp));
        console.log("pool balance of weth after whale:", weth.balanceOf
```

```
            (address(pool)));
34          console.log("pool balance of poolToken after whale:", poolToken
                .balanceOf(address(pool)));
35          vm.stopPrank();
36          console.log("estimated output value:", estimatedOutput);
37
38          vm.startPrank(user1);
39          estimatedOutput = pool.getOutputAmountBasedOnInput(
40              10, //input 10 tokens
41              20, // allowing 20 tokens for reserve
42              10
43          );
44          console.log("user1 PoolToken balance before swap:", poolToken.
                balanceOf(address(user1)));
45          console.log("user1 weth balance before swap:", weth.balanceOf(
                address(user1)));
46          poolToken.approve(address(pool), 100e18);
47          pool.swapExactOutput(poolToken, weth, 3, uint64(block.timestamp
                ));
48          console.log("user1 PoolToken balance after swap:", poolToken.
                balanceOf(address(user1)));
49          console.log("user1 weth balance after swap:", weth.balanceOf(
                address(user1)));
50          vm.stopPrank();
51      }
```

PoC

Recommended Mitigation: We should use a `maxInputAmount` to ensure users do not get charged with more tokens than they are willing to pay.

```
1       function swapExactOutput(
2           IERC20 inputToken,
3           IERC20 outputToken,
4   +       uint256 maxInputAmount,
5           uint256 outputAmount,
6           uint64 deadline
7       )
8           public
9           revertIfZero(outputAmount)
10          revertIfDeadlinePassed(deadline)
11          returns (uint256 inputAmount)
12      {
```

[H-3] `TSwapPool::sellPoolTokens` calls the `TSwapPool::swapExactOutput` function with the incorrect parameter input, causing the intended functionality to be wrong.

Description:

The `sellPoolTokens` function accepts a `poolTokenAmount` parameter but subsequently calls the `swapExactOutput` function using this parameter. However, the `swapExactOutput` function expects the amount to be the `outputAmount`, not the input amount.

Impact:

Using `poolTokenAmount` as the input amount to sell and passing it to the `swapExactOutput` function, which expects the `outputAmount`, leads to a mismatch. This means that instead of limiting the number of tokens we intend to sell, we are setting the actual output amount. Consequently, the limit on how many tokens to sell is not enforced as expected.

Proof of Concept: 1. liquidityProvider funds the pool, we have a whale who mints `poolToken` and sells them for `weth`. We do this to ensure that the conversion ratio is different. 2. We now have a seller who comes along and wants to sell 10 `poolToken` for some `weth` using the `sellPoolTokens` function. 3. The seller is now expecting their `poolToken` balance to be 10 less than what they had before. 4. The sellers balances for `poolToken` and `weth` are incorrect. The seller has 10 `weth` and used up 111 `poolToken` instead of the expected 10 we put into the parameter.

```
Seller weth Balance before selling: 0
Seller poolToken Balance before selling: 200
Seller weth Balance after selling: 10
Seller poolToken Balance after selling: 89
```

```
    function testWrongParamOnSellPoolTokens() public {
        // LP funds pool
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100 ether);
        poolToken.approve(address(pool), 100 ether);
        pool.deposit(100 ether, 100 ether, 100 ether, uint64(block.
            timestamp));
        vm.stopPrank();
        // whale comes to sell alot of poolTokens changing the
            conversion between pooltokens and weth.
        address whale = makeAddr("whale");
        poolToken.mint(address(whale), 100 ether);
        vm.startPrank(whale);
        poolToken.approve(address(pool), 100 ether);
        pool.sellPoolTokens(1 ether);
        vm.stopPrank();
        // new user comes along called seller;
        address seller = makeAddr("seller");
        poolToken.mint(address(seller), 200);
        // check the sellers balances before.
        console.log("Seller weth Balance before selling", weth.
            balanceOf(address(seller)));
        console.log("Seller poolToken Balance before selling",
            poolToken.balanceOf(address(seller)));
        // seller will now sell some pooltokens for weth tokens.
```

```
22          vm.startPrank(seller);
23          poolToken.approve(address(pool), 100 ether);
24          pool.sellPoolTokens(10);
25          vm.stopPrank();
26          // check balances of seller after the sell
27          console.log("Seller weth Balance after selling", weth.balanceOf
                (address(seller)));
28          console.log("Seller poolToken Balance after selling", poolToken
                .balanceOf(address(seller)));
29      }
```

PoC

Recommended Mitigation: It would be best to change the called function from `swapExactOutput` to `swapExactInput`, with this in mind we are going to need to add in another parameter to the function to allow for the function call to be successful. The additional parameter in the function will need to specify the `minWethOutputAmount`.

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

[H-4] Break in invariant for swapping 10 times will give user incentive tokens, causing a malicious user to swap back and forth between two coins draining the funds from the protocol.

Description: The protocol has an incentive that will transfer an extra token once a user has done 10 swaps, this is a break and can cause the funds to be drained by swapping back and forth between two tokens basically not loosing any money and receiving a free token after each 10 swaps.

Impact: This break can give a malicious user the opportunity to do countless swaps with minimum money and draining the funds of the protocol. code with error below:

```
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           }
```

Proof of Concept: Please see the PoC below: 1. We have a user use the `swapExactOutput` function

10 times to simulate 10 total swaps. 2. We see that the pool Balance of Weth is actually missing a token due to the incentive of the protocol.

```
VM::assertEq(-1100000000000000000 [-1.1e18], -1000000000000000000 [-1e17
    ])
```

PoC

```
    function testInvariantBreak() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100 ether);
        poolToken.approve(address(pool), 100 ether);
        pool.deposit(100 ether, 100 ether, 100 ether, uint64(block.
            timestamp));
        vm.stopPrank();

        uint256 outputWethAmount = 1e17;
        poolToken.mint(address(user), 100 ether);

        vm.startPrank(user);
        poolToken.approve(address(pool), 100e18);
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));

        int256 starting_y = int256(weth.balanceOf(address(pool)));
        int256 expectedDelta_y = int256(-1) * int256(outputWethAmount);
        pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(
            block.timestamp));

        vm.stopPrank();

        int256 ending_y = int256(weth.balanceOf(address(pool)));
        int256 actualDelta_y = int256(ending_y) - int256(starting_y);

        assertEq(actualDelta_y, expectedDelta_y);
```

```
33      }
```

Recommended Mitigation: Recommend getting rid of this feature, or allow for the incentive amount to be allocated within a certain amount of fee's collected from user or utilise the fee set up to be the same for the incentive tokens.

```
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 still be processed even after the deadline block.

Description: `uin64 deadline` has not been used, this can cause a user to call for a different `block.timestamp` so that they can still deposit even after it is expected to revert due to time `deadline`. This is told by the natspec "The deadline for the transaction to be completed by", having this param not being used can cause some unfavourable market conditions.

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

Proof of Concept: The `deadline` parameter is unused.

Recommended Mitigation: Add in the following control into the deposit function.

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

Lows

[L-1] `TSwapPool::_addLiquidityMintAndTransfer` emitting parameters in wrong order according to `LiquidityAdded` event, creating a mix up in values.

Description: `TSwapPool::_addLiquidityMintAndTransfer` function is emitting an event called `LiquidityAdded` but the `_addLiquidityMintAndTransfer` function is passing the values in order (`msg.sender, poolTokensToDeposit, wethToDeposit`) but the event order is (`address indexed liquidityProvider, uint256 wethDeposited, uint256 poolTokensDeposited`) giving the event logs the wrong values.

Impact: Emitting the wrong values to an event will cause vlaues to get mixed up and cause confusion once checking the events and their values.

Recommended Mitigation: it is recommended to swap the order to follow the events order.

```
1      function _addLiquidityMintAndTransfer(
2              uint256 wethToDeposit,
3              uint256 poolTokensToDeposit,
4              uint256 liquidityTokensToMint
5          )
6              private
7          {
8              _mint(msg.sender, liquidityTokensToMint);
9  +           emit LiquidityAdded(msg.sender, wethToDeposit,
       poolTokensToDeposit);
10 -           emit LiquidityAdded(msg.sender, poolTokensToDeposit,
       wethToDeposit);
11
12             i_wethToken.safeTransferFrom(msg.sender, address(this),
                   wethToDeposit);
13             i_poolToken.safeTransferFrom(msg.sender, address(this),
                   poolTokensToDeposit);
14         }
```

[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. while it declares the named return value `output` it is never asigned a value or given an explicit return statement.

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

Proof of Concept: Please seek the testing code provided below. 1. We have the liquidityProvider put into the pool so we know there is tokens in their to do a swap effectively, make a new address

called `user1` and have this user mint poolTokens. 2. We console.log the balance to check how much was minted and ensure it hit the `user1` address, `user1` then approves poolTokens ready for swap. 3. `user1` now calls `TSwapPool::swapExactInput()` function with swapping 10 poolTokens for weth tokens, while assigning the function to `uint256 output` to get the return value after the transaction has processed. 4. `user1` has successfully completed the transaction but our output amount shows that there are 0 tokens returns in the console.log

```
    function testSwapExactInputReturnValue() public {
        // need to have a user do a swap and call the return value to see what the balance of bought tokens are.
        // LP deposits into pool.
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100 ether, 100 ether, 100 ether, uint64(block.timestamp));
        vm.stopPrank();

        // brand new user comes in, mints tokens and then calls `swapExactInput` function.
        address user1 = makeAddr('user1');
        // user mints tokens
        poolToken.mint(address(user1), 1e3);
        // retreieve the balance after minting tokens
        console.log(poolToken.balanceOf(user1));
        // user1 now wants to swap some poolToken to get some weth.
        vm.startPrank(user1);
        //user1 approves tokens to transfer to pool contract address.
        poolToken.approve(address(pool),100e18);
        // attach the return value to output so that we can view the result.
        uint256 output = pool.swapExactInput(
            poolToken,
            10,
            weth,
            1,
            uint64(block.timestamp)
        );
        console.log("Returned output amount after swap:", output);
            vm.stopPrank();
    }
```

console.log Results:

```
    user1 poolToken Balance: 1000
    Returned output amount after swap: 0
```

PoC

Recommended Mitigation:

```
    function swapExactInput (
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
            uint256 output
        )
    {
        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(output, minOutputAmount);
        }

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

Informationals

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

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

[I-2] Lacking zero address checks

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

```
4 +         }
5           i_wethToken = wethToken;
6       }
```

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

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

[I-4] Unnecessary use of numeric values, consider using scientific notation instead.

```
1 -         uint256 private constant MINIMUM_WETH_LIQUIDITY = 1_000_000_000
    ;
2 +         uint256 private constant MINIMUM_WETH_LIQUIDITY = 1e9;
```

[I-5] `TSwapPool::deposit` unused `poolTokenReserves` creating Unnecessary extra costs of gas and should be removed if unused.

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

[I-6] `TSwapPool::deposit` does not follow CEI formula and should.

```
1 +         liquidityTokensToMint = wethToDeposit;
2           _addLiquidityMintAndTransfer(wethToDeposit,
                maximumPoolTokensToDeposit, wethToDeposit);
3 -         liquidityTokensToMint = wethToDeposit;
```

[I-7] Use of magic numbers in `TSwapPool::getOutputAmountBasedOnInput` and should consider using constant varibles.

Description: in the function `TSwapPool::getOutputAmountBasedOnInput` there are uses of magic numbers or hard coded numbers in each function, it can be easily overlooked or mistaken with difference values, this is why we recommend using constant variables ensuring all numbers are expected.

```
+       uint256 FEE_PERCENTAGE_MULTIPLIER = 997;
+       uint256 FULL_AMOUNT_MULTIPLIER = 1000;

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