Protocol Audit Report

Version 1.0

Cyfrin.io

June 23, 2024

Protocol Audit Report

Mudit Jain

June 23, 2024

Prepared by: Mudit Jain

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

TSwap Pools

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

For example: 1. User A has 10 USDC 2. They want to use it to buy DAI 3. They `swap` their 10 USDC -> WETH in the USDC/WETH pool 4. Then they `swap` their WETH -> DAI in the DAI/WETH pool

Every pool is a pair of `TOKEN X` & `WETH`.

There are 2 functions users can call to swap tokens in the pool. - `swapExactInput` - `swapExactOutput`

We will talk about what those do in a little.

Liquidity Providers

In order for the system to work, users have to provide liquidity, aka, “add tokens into the pool”.

Why would I want to add tokens to the pool?

The TSwap protocol accrues fees from users who make swaps. Every swap has a `0.3` fee, represented in `getInputAmountBasedOnOutput` and `getOutputAmountBasedOnInput`. Each applies a `997` out of `1000` multiplier. That fee stays in the protocol.

When you deposit tokens into the protocol, you are rewarded with an LP token. You’ll notice `TSwapPool` inherits the `ERC20` contract. This is because the `TSwapPool` gives out an ERC20 when Liquidity Providers (LP)s deposit tokens. This represents their share of the pool, how much they put in. When users swap funds, 0.03% of the swap stays in the pool, netting LPs a small profit.

LP Example

1. LP A adds 1,000 WETH & 1,000 USDC to the USDC/WETH pool
    1. They gain 1,000 LP tokens
2. LP B adds 500 WETH & 500 USDC to the USDC/WETH pool
    1. They gain 500 LP tokens
3. There are now 1,500 WETH & 1,500 USDC in the pool
4. User A swaps 100 USDC -> 100 WETH.
    1. The pool takes 0.3%, aka 0.3 USDC.
    2. The pool balance is now 1,400.3 WETH & 1,600 USDC
    3. aka: They send the pool 100 USDC, and the pool sends them 99.7 WETH

Note, in practice, the pool would have slightly different values than 1,400.3 WETH & 1,600 USDC due to the math below.

Core Invariant

Our system works because the ratio of Token A & WETH will always stay the same. Well, for the most part. Since we add fees, our invariant technially increases.

`x * y = k` - x = Token Balance X - y = Token Balance Y - k = The constant ratio between X & Y

Our protocol should always follow this invariant in order to keep swapping correctly!

Make a swap

After a pool has liquidity, there are 2 functions users can call to swap tokens in the pool. - `swapExactInput` - `swapExactOutput`

A user can either choose exactly how much to input (ie: I want to use 10 USDC to get however much WETH the market says it is), or they can choose exactly how much they want to get out (ie: I want to get 10 WETH from however much USDC the market says it is.

Disclaimer

Mudit 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	2	H/M	M
	Medium	0	M	M/L
	Low	1	M/L	L
	Informational	4	M/L	L
	Gas	0	M/L	L

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
```

Compatibilities

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum

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

High

[H-1] `TSwapPool::deposit` function does not consider deadline of transaction provided. Transaction can be called even after deadline

Description: The `TSwapPool::deposit` function provides user to define the deadline of the transaction. But it does not consider the time of transaction. So the transaction can be processed anytime even after deadline.

Impact: The transaction can be halted by malicious miners and be executed after deadline when the prices in the DEX are not favourable to the users and profitable for miners

Recommended Mitigation: 1. Put a check in the function to revert the transaction if deadline has passed.

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
```

```
7           external
8 +         revertIfDeadlinePassed(deadline)
9           revertIfZero(wethToDeposit)
10          returns (uint256 liquidityTokensToMint)
11      {
12  .
13  .
14  .
15      }
```

[H-2] `TSwapPool::getInputAmountBasedOnOutput` gives wrong answer and charges high amount of fee.

Description: The `TSwapPool::getInputAmountBasedOnOutput` function wrongly calculates fees and estimates the inputAmount required to get the output. The precision is incorrectly set to 1_000 instead of 100.

Impact: The users would be wrongly charged high inputAmount for some swap which may discourage users to use the TSwap

Proof of Concept:

POC

In the tests contract for TSwapPool paste the following lines

```
1       function test_swapExactInput_ChargesHighFees() public {
2           // input Amount should be (inputReserves*outputAmount)/(
                outputReserves - outputAmount) + fees on exchange (0.3% fees
                 on output amount)
3           uint256 inputReserves = 1e18 ;
4           uint256 outputAmount = 2e18 ;
5           uint256 outputReserves = 3e18;
6           uint256 expectedInputAmount = ((inputReserves * outputAmount) *
                1000) /
7               ((outputReserves - outputAmount) * 997);
8           uint256 actualInputAmount = pool.getInputAmountBasedOnOutput(
                outputAmount, inputReserves, outputReserves)
9           assertEq(inputAmount, actualInputAmount);
10      }
```

Recommended Mitigation: 1. Avoid use of magic numbers for precision and assign constant values in beginning of contract.

```
1       IERC20 private immutable i_wethToken;
2       IERC20 private immutable i_poolToken;
3       uint256 private constant MINIMUM_WETH_LIQUIDITY = 1_000_000_000;
4       uint256 private swap_count = 0;
```

```
5       uint256 private constant SWAP_COUNT_MAX = 10;
6   +   uint256 private constant FEE_PRECISION = 1000;
7   .
8   .
9   .
10      function getInputAmountBasedOnOutput(
11          uint256 outputAmount,
12          uint256 inputReserves,
13          uint256 outputReserves
14      )
15          public
16          pure
17          revertIfZero(outputAmount)
18          revertIfZero(outputReserves)
19          returns (uint256 inputAmount)
20      {
21  -       return ((inputReserves * outputAmount) * 10000) / ((
        outputReserves - outputAmount) * 997);
22  +       return ((inputReserves * outputAmount) * FEE_PRECISION) / ((
        outputReserves - outputAmount) * 997);
23      }
```

Low

[L-1] Wrong order of variables emitted in the `TSwapPool::LiquidityAdded` event emission

Description: The `TSwapPool::LiquidityAdded` event should be emitted with the fields in order (liquidityProviderAddress, wethDeposited, poolTokensDeposited) but in the `TSwapPool::_addLiquidityMintAndTransfer` function the order of variables has been wrong.

Impact: Wrong Information emitted from events causing websites searching for these events getting wrong information

Recommended Mitigation: 1. Consider changing order of variables emitted in the event.

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

```
11          i_poolToken.safeTransferFrom(
12              msg.sender,
13              address(this),
14              poolTokensToDeposit
15          );
16      }
```

Informational

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error not used anywhere in the codebase

Description: The error `PoolFactory::PoolFactory__PoolDoesNotExist` is not used.

Recommended Mitigation: 1. remove the error from the contract.

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

[I-2] Cosnstructor in `PoolFactory` lacks zero check

Description: `PoolFactory::constructor` lacks zero check before assigning value to `PoolFactory::i_wethToken`

Recommended Mitigation: 1. Add zero check before assignment.

```
1       constructor(address wethToken) {
2   +       require(wethToken != address(0), "Zero Address assignment");
3           i_wethToken = wethToken;
4       }
```

[I-3] `PoolFactory::createPool` assigns name to the symbol of deployed ERC20 instead of symbol

Description: `PoolFactory::createPool` assigs liquidity token symbol as concatenation of “ts” and symbol of token in the DEX.

Recommended Mitigation:

```
1       function createPool(address tokenAddress) external returns (address
            ) {
2           if (s_pools[tokenAddress] != address(0)) {
3               revert PoolFactory__PoolAlreadyExists(tokenAddress);
```

```
4           }
5           string memory liquidityTokenName = string.concat("T-Swap ",
                IERC20(tokenAddress).name());
6   -       string memory liquidityTokenSymbol = string.concat("ts", IERC20
        (tokenAddress).name());
7   +       string memory liquidityTokenSymbol = string.concat("ts", IERC20
        (tokenAddress).symbol());
8           TSwapPool tPool = new TSwapPool(tokenAddress, i_wethToken,
                liquidityTokenName, liquidityTokenSymbol);
9           s_pools[tokenAddress] = address(tPool);
10          s_tokens[address(tPool)] = tokenAddress;
11          emit PoolCreated(tokenAddress, address(tPool));
12          return address(tPool);
13      }
```

[I-4] unused variables in `TSwapPool::deposit` function

Description: The `poolTokenReserver` is declared but not used anywhere in `TSwapPool::deposit` function.

Recommended Mitigation: 1. Remove the variable from the function.

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