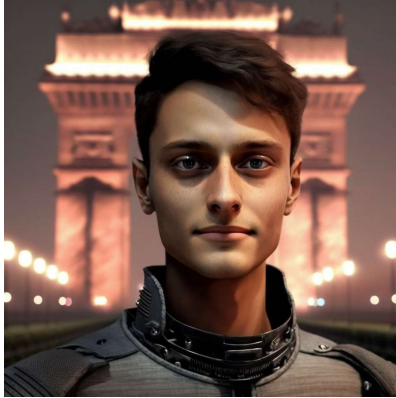


TSwap Protocol Audit Report

0xAdra

April 05, 2024



TSwap Protocol Audit Report

Version 1.0

0xAdra

April 5, 2024

TSwap Protocol Audit Report

0xAdra

April 05, 2024

Prepared by: 0xAdra

Lead Auditors: 0xAdra

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
 - Scope
 - Roles
 - Issues found
- Findings
- High
- Medium
- Low
- Informational

Protocol Summary

This protocol 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

0xAdra 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		
Likelihood	High	High	Medium	Low
	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

Scope

./src/PoolFactory.sol

./src/TSwapPool.sol

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

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

Sevurity	Number of issues found
High	4
Medium	2
Low	2
Info	4
Total	12

Findings

High

[H-1] Incorrect fee calculation in TSwapPool::getInputAmountBasedOnOutput casuses protocol to take to many tokens from the users, resulting in lost fees

Description: The getInputAmountBasedOnOutput funtion is intended to calculate the amount of tokens a user should given an amount of tokens of output tokens. However, the function miscalcuates the resulting amount . When calculating the fee it scale the amount by 10_000 instead of 1_000.

Impact: Protocol takes more fees than expected from users.

Proof of Code : To test this, include the following code in the TSwapPool.t.sol file:

```
function testFlawedSwapExactOutput() public {
    uint256 initialLiquidity = 100e18;
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), initialLiquidity);
    poolToken.approve(address(pool), initialLiquidity);

    pool.deposit({
        wethToDeposit: initialLiquidity,
        minimumLiquidityTokensToMint: 0,
        maximumPoolTokensToDeposit: initialLiquidity,
        deadline: uint64(block.timestamp)
    });
    vm.stopPrank();

    // User has 11 pool tokens
    address someUser = makeAddr("someUser");
    uint256 userInitialPoolTokenBalance = 11e18;
    poolToken.mint(someUser, userInitialPoolTokenBalance);
    vm.startPrank(someUser);

    // Users buys 1 WEIH from the pool, paying with pool tokens
    poolToken.approve(address(pool), type(uint256).max);
    pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.timestamp));

    // Initial liquidity was 1:1, so user should have paid ~1 pool token
    // However, it spent much more than that. The user started with 11 token
    assertLt(poolToken.balanceOf(someUser), 1 ether);
    vm.stopPrank();
}
```

```

// The liquidity provider can rug all funds from the pool now,
// including those deposited by user.
vm.startPrank(liquidityProvider);
pool.withdraw(
    pool.balanceOf(liquidityProvider),
    1, // minWethToWithdraw
    1, // minPoolTokensToWithdraw
    uint64(block.timestamp)
);

assertEq(weth.balanceOf(address(pool)), 0);
assertEq(poolToken.balanceOf(address(pool)), 0);
}

```

Recommended Mitigation: Consider the following :

```

function getInputAmountBasedOnOutput(
    uint256 outputAmount,
    uint256 inputReserves,
    uint256 outputReserves
)
    public
    pure
    revertIfZero(outputAmount)
    revertIfZero(outputReserves)
    returns (uint256 inputAmount)
{
-     return ((inputReserves * outputAmount) * 10000)/((outputReserves - outputAmount));
+     return ((inputReserves * outputAmount) * 1000)/((outputReserves - outputAmount));
}

```

[H-2] Lack of slippage protection in TSwapPool::swapExactOutput casues users to potencially receive way fewer tokens.

Description: The swapExactOutput function does not include any sort of slippage protection. thi sfunction similar to what i sdone in swapExactInput,where the function specifies a minOutputAmount, the swapExactOutput function should specify a maxInputAmount.

Impact: If markets conditions change before the txns processes the users can get a much more worse swap.

Proof of Concept:

1.The price of WETH is 1,000 USDC 2. User inputs swapExactOutput looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline does nothing 3. The function does not offer a maxInput amount 4.

As the txn 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. 5. The txn completes but the user sent the protocol 10,000 USDC instead of 1,000 USDC.

Recommended Mitigation: Consider the following changes, We should include the maxInputAmount so the user has to spend up to a specific amount , and can predict how much they will spend on the protocol.

```

function swapExactOutput(
    IERC20 inputToken ,
+    uint256 maxInputAmount ,
.
.
.
    inputAmount = getInputAmountBasedOnOutput(outputAmount , inputReserves , outputToken , outputAmount)
+    if (inputAmount > maxInputAmount) {
+        revert ();
+    }
    _swap(inputToken , inputAmount , outputToken , outputAmount);

```

[H-3] The TSwapPool::sellPoolTokens mismatches input and output tokens causing users to receive the incorrect amounts of tokens

Description: The sellPoolTokens function intends to allow users to easily sell pool tokens and receive the WETH in exchange. Users indicate how many pool tokens they are willing to sell in the PoolTokenAmount parameter. however, the fn currently miscalculates the swapped amounts.

This is due to the fact that the swapExactOutput function is called instead of swapExactInput, because the users specifies exact amount of input tokens, not output tokens.

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 adding minWethToReceive to the parameter.

```

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

```

}

[H-4] In TSwapPool::_swap the extra tokens given to users after the 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 - 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_swap function. Meaning that over time the protocol funds will be drained.

The following block of code is responsible for the issue.

```
swap_count++;
if (swap_count >= SWAP_COUNT_MAX) {
    swap_count = 0;
    outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
}
```

Impact: A user could maliciously drain the protocol funds by doing a lot of swaps and collecting the extra incentives given out by the protocol.

More simply, the protocols core invariant is broken.

Proof of Concept: 1. A user swaps 10 times, and collects the extra incentives of 1_000_000_000_000_000_000 tokens.

2. the user continues to swap until all the protocol funds are drained.

Proof of Code

Place the following in the TSwapPool.t.sol.

```
function testInvariantBroken() public {
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), 100e18);
    poolToken.approve(address(pool), 100e18);
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
    vm.stopPrank();

    uint256 outputWeth = 1e17;

    vm.startPrank(user);
    poolToken.approve(address(pool), type(uint64).max);
    poolToken.mint(user, 100e18);
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
}
```



```

pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));

int256 startingY = int256(poolToken.balanceOf(address(pool)));
int256 expectedDeltaY = int256(-1) * int256(outputWeth);

pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
vm.stopPrank();

uint256 endingY = poolToken.balanceOf(address(pool));
int256 actualDeltaY = int256(endingY) - int256(startingY);
assertEq(actualDeltaY, expectedDeltaY);
}

```

Recommended Mitigation: Remove the incentive mechanism. If u 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.

```

swap_count++;
if (swap_count >= SWAP_COUNT_MAX) {
    swap_count = 0;
    outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
}

```

Medium

[M-1] TSwapPool::deposit is missing deadline checks causing txns to complete even after the deadline

Description: the deposit fn accepts a deadline parameter, which acc to the docs is “the deeadline 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 unfavourable.

<!-- MEV attacks -->

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

Proof of Concept: The deadline parameter is unused.

Recommended Mitigation: Consider making the following changes to function.

```
function deposit(  
    uint256 wethToDeposit ,  
    uint256 minimumLiquidityTokensToMint , /* LP tokens - if empty, we can p  
    uint256 maximumPoolTokensToDeposit ,  
    uint64 deadline  
)  
    external  
+    revertIfDeadlinePassed (deadline)  
    revertIfZero (wethToDeposit)  
    returns (uint256 liquidityTokensToMint)  
{
```

[M-2] Rebase fee-on-transfer , ERC-777 & centralized ERC20s break the protocol invariant.

Description: The core protocol invariant is : $x * y = k$. In practice, the protocol takes fees and actually increase k . so we need to make sure $x * y = k$ before the fees are applied.

Impact:

Proof of Concept:

Recommended Mitigation: Recheck the protocol invariant.

Low

[L-1] the TSwapPool::LiquidityAdded event has parameter out of order.

Description: When the LiquidityAdded event is emitted in __addLiquidityMintAndTransfer function it logs values in an incorrect order. The poolTokensToDeposit value should go in the 3rd parameter position & wethToDeposit in 2nd parameter.

Impact: Event emission is incorrect, leading to off chain function potentially malfunctioning.

Recommended Mitigation: Consider making the following changes

```
-    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 fn is expected to return the actual amount of token bought by caller. However , while it declares the named return values

output it is never assigned a value, nor uses an explicit return statement.

Impact: The return value will also be 0, giving incorrect info to the caller

Proof of Concept:

Recommended Mitigation: Consider the following changes :

```
-      uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount, inputRes
+      output = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputRes

-      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);
```

Informational

[I-1] PoolFactory__PoolDoesNotExist is not used should be removed from PoolFactory::PoolFactory__PoolDoesNotExist.

Description:

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

[I-2] Lacking zero address check in PoolFactory constructor.

Description:

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

[I-3] PoolFactory::liquidityTokenSymbol should use .symbol() not .name()

Description:

```

- string memory liquidityTokenSymbol = string.concat("ts",IERC20(tokenAddress)).
+ string memory liquidityTokenSymbol = string.concat("ts",IERC20(tokenAddress)).

```

[I-4]: 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.

- Found in src/PoolFactory.sol Line: 35

```
event PoolCreated(address tokenAddress, address poolAddress);
```

- 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(
```