TSwap Protocol Audit Report

Version 1.0

BLaeir

February 8, 2024

TSwap Protocol Audit Report

BLaeir

08/02/2024

Prepared by: BLaeir Lead Security Researcher: - BLaeir

Table of Contents

    * [H-5] The `TSwapPool::_swap` token incentives for swaps breaks the protocols invariant of `x * y = k`
  - Medium
    * [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline
    * [M-2] Using `ERC721::_mint()` can be dangerous
  - Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters that are out of order
    * [L-2] `TSwapPool::swapExactInput` is giving the wrong return value and will always return zero
    * [L-3] PUSH0 is not supported by all chains
  - Informationals
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is never used and should be removed
    * [I-2] `PoolFactory::wethToken` is lacking a zero address check
    * [I-3] `PoolFactory::liquidityTokenSymbol` should use `.symbol()` and not `.name()` to differiantate between the two strings
    * [I-4] `TSwapPool` Event is missing `indexed` fields
    * [I-5] `TSwapPool::wethToken` is lacking a zero address check
    * [I-6] `TSwapPool::Deposit` doesn't follow CEI, recommend having `_addLiquidityMintAndTr` call at end
    * [I-7] Use of magic numbers are discouraged
    * [I-8] `TSwapPool::swapExactInput` Is missing natspec
    * [I-9] It can be confusing to see number literals in a codebase, and it's much more readable if numbers are given a name
    * [I-10] Use of magic numbers are discouraged
  - Gas
    * [G-1] `TSwapPool::deposit` has a line that waste gas

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

The BLa 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
	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

Roles

- Liquidity provider: Can add and remove liquidity from pools
- Swapper/User: Can make swaps using the pools

Executive Summary

Add some notes about how the audit went, types of things found and etc.

We spent X hours with Z auditors using Y tools. etc

Issues found

Severity	Number of issues found
High	5
Medium	2
Low	3
Info	11
Total	21

Findings

High

[H-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, operationrs 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 to deposit, even 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, // LP tokens -> if empty,
             we can pick 100% (100% == 17 tokens)
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[H-2] `TSwapPool::getInputAmountBasedOnOutput` has a miscalculation in it's fees causing loss of tokens from users and loss of fees for the protocol.

Description: The `getInputAmountBasedOnOutput` function miscalculates the fees, so instead of the number being 1,000 that it's supposed to be it has been set to 10,000 causing the input and output token amount for a user to be wrong.

Impact: The protocol takes a bigger cut than intended

Proof of Concept: (In a ocmpetitive audit you'd defos need this)

Recommended Mitigation:

```
function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10000) / ((
    outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1000) / ((
    outputReserves - outputAmount) * 997);
    }
```

[H-3] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens

Description: The `swapExactOutput` function does not include any slippage protection. This function is similar to what is done in `swapExactInput`, where the function specifies a `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`.

Impact: Sandwhich, MEV attacks, unfavourable swap at the time the transaction takes place resulting in loss of tokens when swapping

Proof of Concept: 1. The price of 1 WETH right now is 1,000 USDC 2. User inputs a swapExactOutput looking for 1 WETH inputToken = USDC outputToken = WETH outputAmount = 1 deadline = whatever 3. The function does not offer a maxInput amount 4. 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 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 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, outputReserves);
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-4] `TSwapPool::sellPoolTokens` mixed input and output tokens causing the users to recieve wrong amount of tokens

Description: The `sellPoolToken` function is intended to allow users to easily sell pool tokens and recieve 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`. Doing this would also require a change for the `sellPoolTokens` function to accept a new parameter (ie `minWethToRecieve` to be passed to `swapExactInput`)

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

[H-5] The `TSwapPool::_swap` token incentives for swaps breaks the protocols invariant of `x * y = k`

Description: The protocol follows a strict invariant of `x * y = k`. Where: - `x`: The balance of the protocol 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 th extra incentive in the `_swap` function. Meaning that over the time the protocol funds will be drained

The code 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 of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol. Simlpy put, the protocols core invariant is broken.

Proof of Concept: 1. A user swaps 10 times, and collects the extra incentive 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`

```
function testInvariantSwapBreak() 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(uint256).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(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

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

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

Recommended Mitigation: Remove the incentive mechanism. if you want to keep it then you should think about changing the x * y = k protocol invariant. Or tokens should be set aside the same way as done with the 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 check causing transactions to complete even after the deadline

Description: The `deposit` function accepts a deadline parameter, which according to the documentation is “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 random times, in market conditions where the deposit rate is not unfavourable

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: Consider making the following change to the function.

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[M-2] Using `ERC721::_mint()` can be dangerous

Using `ERC721::_mint()` can mint ERC721 tokens to addresses which don’t support ERC721 tokens. Use `_safeMint()` instead of `_mint()` for ERC721.

- Found in src/TSwapPool.sol Line: 195

  ```
          _mint(msg.sender, liquidityTokensToMint);
  ```

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters that are out of order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran` 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 failing.

Recommended Mitigation:

```
-   emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+   emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit);
```

[L-2] `TSwapPool::swapExactInput` is giving the wrong return value and will always return zero

Description: The `swapExactInput` function is expected to return the amount of tokens bought by the caller. However, while it declares the return value `output` it is never given a value/returns a statement

Impact:

Proof of Concept:

Recommended Mitigation:

```
{
        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);
      }
```

[L-3] PUSH0 is not supported by all chains

Description: 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.

- Found in src/PoolFactory.sol Line: 15

  ```
  pragma solidity 0.8.20;
  ```

- Found in src/TSwapPool.sol Line: 15

  ```
  pragma solidity 0.8.20;
  ```

Informationals

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is never used and should be removed

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

[I-2] `PoolFactory::wethToken` is lacking a zero address check

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

[I-3] `PoolFactory::liquidityTokenSymbol` should use `.symbol()` and not `.name()` to differiantate between the two strings

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

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

[I-4] `TSwapPool` 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(
  ```

[I-5] `TSwapPool::wethToken` is lacking a zero address check

```
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    )
        ERC20(liquidityTokenName, liquidityTokenSymbol)
    {
+           if (wethToken == address(0)) {
+             revert();
        }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

[I-6] `TSwapPool::Deposit` doesn't follow CEI, recommend having `_addLiquidityMintAndTransfer` call at end

```
else {
+             liquidityTokensToMint = wethToDeposit;
             _addLiquidityMintAndTransfer(wethToDeposit,
                maximumPoolTokensToDeposit, wethToDeposit);
-             liquidityTokensToMint = wethToDeposit;
        }
```

[I-7] Use of magic numbers are discouraged

It can be confusing to see number literals in a codebase, and it's much more readable if numbers are given a name and then used as it is

~~~
Examples:

```javascript
    uint256 inputAmountMinusFee = inputAmount * 997;
    uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
```
Better doing something like this

```javascript
uint256 public constant INPUT_FEE = 997
uint256 constant FEE_ADJUSTMENT_FACTOR = 1000;
```
~~~

[I-8] `TSwapPool::swapExactInput` Is missing natspec

[I-9] It can be confusing to see number literals in a codebase, and it's much more readable if numbers are given a name

~~~
Examples:

```javascript
    outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
```
Better doing something like this, although this breaks protocol
   invariant so expecting the whole _swap function to be redone

```javascript
uint256 public constant ONE_ETHER = 1_000_000_000_000_000_000
```

~~~

~~~
```javascript
    outputToken.safeTransfer(msg.sender, ONE_ETHER);
```
~~~

[I-10] Use of magic numbers are discouraged

It can be confusing to see numbers literals in a codebase, and it's much more readable if numbers are given a name. This goes for both of these functions in the example below

~~~
Examples:

```javascript
    function getPriceOfOneWethInPoolTokens() external view returns (
        uint256) {
    return
    getOutputAmountBasedOnInput(1e18, i_wethToken.balanceOf(address(
        this)), i_poolToken.balanceOf(address(this)));
}

function getPriceOfOnePoolTokenInWeth() external view returns (uint256)
     {
    return
    getOutputAmountBasedOnInput(1e18, i_poolToken.balanceOf(address(
        this)), i_wethToken.balanceOf(address(this)));
}
```
Better doing something like this

```javascript
uint256 public constant STANDARD_INPUT_AMOUNT = 1e18
```

```javascript
    function getPriceOfOneWethInPoolTokens() external view returns (
        uint256) {
    return
    getOutputAmountBasedOnInput(STANDARD_INPUT_AMOUNT, i_wethToken.
        balanceOf(address(this)), i_poolToken.balanceOf(address(this)));
}

function getPriceOfOnePoolTokenInWeth() external view returns (uint256)
     {
    return
    getOutputAmountBasedOnInput(STANDARD_INPUT_AMOUNT, i_poolToken.
        balanceOf(address(this)), i_wethToken.balanceOf(address(this)));
}
```
~~~

Gas

[G-1] `TSwapPool::deposit` has a line that waste gas