TSwap Protocol Audit Report

Version 1.0

mahithchigurupati.me

June 23, 2024

TSwap Protocol Audit Report

mahithchigurupati.me

June 22, 2024

Prepared by: Mahith

Lead Auditors: - Mahith

Table of Contents

- High
  - [H-1] `TSwapPool::getInputAmountBasedOnOutput()` calculates pool fee as a wrong value, thereby taking more tokens than intended from caller
  - [H-2] `TSwapPool::_swap()` is giving away extra tokens, breaking system's invariant property
  - [H-3] `sellPoolTokens` is calculating w.r.to output instead of input
  - [H-4] `TSwapPool::swapExactOutput()` function is missing slippage protection check, causing caller to get less tokens than they expect
  - [H-5] `TSwapPool::deposit()` doesn't take `deadline` parameter into consideration, causing depositors to get unexpected lp token value for their deposit
- Medium
  - [M-1] weird-ERC20, ERC777 can break protocol invariant
- Low
  - [L-1] `PoolFactory()::constructor()` must have a zero check, to avoid pool creation with `address(0)`
  - [L-2] Incorrect parameter logs in `TSwapPool::LiquidityAdded` event
  - [L-3] `swapExactInput` doesn't return expected `output` value
- Informational/ Non-critical
  - [I-1] Test Coverage
  - [I-2] **`public`** functions not used internally could be marked `external`
  - [I-3] Define and use `constant` variables instead of using literals
  - [I-4] Event is missing `indexed` fields
  - [I-5] PUSH0 is not supported by all chains
  - [I-6] Large literal values multiples of 10000 can be replaced with scientific notation
  - [I-7] Unused Custom Error
  - [I-8] Follow CEI

Protocol Summary

TSwap

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.

This codebase is based loosely on Uniswap v1

Disclaimer

Mahith 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

NA

Scope

- Commit Hash: XXX
- 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 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

NA

Issues found

Severity	Number of issues found
High	5
Medium	1
Low	3
Info	8
Total	17

Findings

High

[H-1] `TSwapPool::getInputAmountBasedOnOutput()` calculates pool fee as a wrong value, thereby taking more tokens than intended from caller

Description:

`TSwapPool::swapExactOutput` calls `TSwapPool::getInputAmountBasedOnOutput` to get input amount to supply based on output amount expected, but the function `getInputAmountBasedOnOutput` calculate fee with an error. The actual fee expected by the protocol is 0.3% of the swap amount requested. But, this function is calculating fee as 90.3% of the swap thereby taking away more amount than user expects.

Impact: user loses 90% more tokens as fee than what protocol says i.e., 0.3% thereby user lose of funds for user.

Proof of Concept:

Place below code in `Tswap.t.sol` and run `forge test --mt testswapExactOutputIsWrong`

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

        address user1 = makeAddr("user1");
        poolToken.mint(user1, 100e18);

        vm.startPrank(user1);
        poolToken.approve(address(pool), 100e18);

        // what is 0.3% of 1e18 = 3e15
        // so, we need to pay tokenA of 1e18 + 3e15 = 1.003e18 in exchange of 1 weth
        // user1 starts with balance of 100e18. so, after swap, user balance must be -
        // 100e18 - 1.003e18 = 98.997e18

        pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.timestamp));

        // so expected is - 98.997e18, lets see what we got -
        console.log(poolToken.balanceOf(user1));

        // user1 must have greater than 98e18 atleast, but he has less than that -
        assertFalse(poolToken.balanceOf(user1) > 98e18);
    }
```

Recommended Mitigation:

Make below code changes in `TSwapPool.sol`

```
    function getInputAmountBasedOnOutput(
        ...
    )
        ...
    {
-         return ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997);
+         return ((inputReserves * outputAmount) * 1000) / ((
```

```
        outputReserves - outputAmount) * 997);
8
9       }
```

[H-2] `TSwapPool::_swap()` is giving away extra tokens, breaking system's invariant property

Description: for every 10 swaps, there is an additional transfer of 1 ether to the swapper hence, the protocol invariant breaks

Impact: protocol breaks and becomes unusable if invariant breaks.

Proof of Concept:

code

Place below code in `test/invariants/Invariant.t.sol`

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity 0.8.20;
4
5  import { Test, StdInvariant } from "forge-std/Test.sol";
6
7  import { TSwapPool } from "../../src/TSwapPool.sol";
8  import { PoolFactory } from "../../src/PoolFactory.sol";
9  import { ERC20Mock } from "@openzeppelin/contracts/mocks/token/
       ERC20Mock.sol";
10 import { Handler } from "./Handler.sol";
11
12 contract Invariant is StdInvariant, Test {
13     TSwapPool public pool;
14     PoolFactory public factory;
15     ERC20Mock public token;
16     ERC20Mock public weth;
17     Handler public handler;
18
19     address liquidityProvider = makeAddr("liquidityProvider");
20
21     uint256 public STARTING_WETH = 100 ether;
22     uint256 public STARTING_TOKEN = 50 ether;
23
24     function setUp() public {
25         factory = new PoolFactory(address(weth));
26
27         weth = new ERC20Mock();
28         token = new ERC20Mock();
29
30         pool = TSwapPool(factory.createPool(address(token)));
```

```
31
32          vm.startPrank(liquidityProvider);
33          weth.mint(liquidityProvider, STARTING_WETH);
34          token.mint(liquidityProvider, STARTING_TOKEN);
35          pool.deposit(STARTING_WETH, STARTING_WETH, STARTING_TOKEN,
                uint64(block.timestamp));
36
37          handler = new Handler(address(pool), address(weth), address(
                token));
38
39          bytes4[] memory selectors = new bytes4[](2);
40          selectors[0] = handler.deposit.selector;
41          selectors[1] = handler.swap.selector;
42
43          targetSelector(FuzzSelector({ addr: address(handler), selectors
                : selectors }));
44          targetContract(address(handler));
45      }
46
47      function stateful_InvariantX() public view {
48          assertEq(handler.actualDeltaWeth(), handler.expectedDeltaWeth()
                );
49      }
50
51      function stateful_InvariantY() public view {
52          assertEq(handler.actualDeltaToken(), handler.expectedDeltaToken
                ());
53      }
54  }
```

Place below code in `test/invariants/Handler.sol`

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.20;
3
4  import { Test } from "forge-std/Test.sol";
5
6  import { TSwapPool } from "../../src/TSwapPool.sol";
7  import { ERC20Mock } from "@openzeppelin/contracts/mocks/token/
       ERC20Mock.sol";
8
9  contract Handler is Test {
10     TSwapPool public pool;
11     ERC20Mock public weth;
12     ERC20Mock public token;
13
14     address liquidityProvider = makeAddr("liquidityProvider");
15     address user = makeAddr("user");
16
17     uint256 public actualDeltaWeth;
18     uint256 public actualDeltaToken;
```

```
19
20      uint256 public expectedDeltaWeth;
21      uint256 public expectedDeltaToken;
22
23      uint256 startingWeth;
24      uint256 startingToken;
25
26      constructor(address _tswapPool, address _weth, address _token) {
27          pool = TSwapPool(_tswapPool);
28          weth = ERC20Mock(_weth);
29          token = ERC20Mock(_token);
30      }
31
32      function deposit(uint256 wethToDeposit) public {
33          bound(wethToDeposit, pool.getMinimumWethDepositAmount(), type(
                uint64).max);
34          uint256 poolTokenToDeposit = pool.
                getPoolTokensToDepositBasedOnWeth(wethToDeposit);
35
36          startingWeth = weth.balanceOf(address(pool));
37          startingToken = token.balanceOf(address(pool));
38
39          expectedDeltaToken = poolTokenToDeposit;
40          expectedDeltaWeth = wethToDeposit;
41
42          vm.startPrank(liquidityProvider);
43          weth.mint(liquidityProvider, wethToDeposit);
44          token.mint(liquidityProvider, poolTokenToDeposit);
45
46          weth.approve(address(pool), wethToDeposit);
47          token.approve(address(pool), poolTokenToDeposit);
48
49          pool.deposit(wethToDeposit, 0, poolTokenToDeposit, uint64(block
                .timestamp));
50          vm.stopPrank();
51
52          actualDeltaWeth = weth.balanceOf(address(pool)) - startingWeth;
53          actualDeltaToken = token.balanceOf(address(pool)) -
                startingToken;
54      }
55
56      function swap(uint256 outputWethAmount) public {
57          bound(outputWethAmount, 0, weth.balanceOf(address(pool)));
58          uint256 inputTokenAmount = pool.getInputAmountBasedOnOutput(
59              outputWethAmount, token.balanceOf(address(pool)), weth.
                    balanceOf(address(pool))
60          );
61
62          startingWeth = weth.balanceOf(address(pool));
63          startingToken = token.balanceOf(address(pool));
64
```

```
65          expectedDeltaWeth = weth.balanceOf(address(pool)) -
               outputWethAmount;
66          expectedDeltaToken = token.balanceOf(address(pool)) +
               inputTokenAmount;
67
68          vm.prank(user);
69          token.mint(user, inputTokenAmount);
70          token.approve(address(pool), inputTokenAmount);
71          pool.swapExactOutput(token, weth, outputWethAmount, uint64(
               block.timestamp));
72          vm.stopPrank();
73
74          actualDeltaWeth = weth.balanceOf(address(pool)) - startingWeth;
75          actualDeltaToken = token.balanceOf(address(pool)) -
               startingToken;
76      }
77  }
```

Now run `forge test --mt stateful_InvariantX` and `forge test --mt stateful_InvariantY` to see if invariant breaks or no.

Recommended Mitigation:

Make below code changes in `TSwapPool.sol`

```
1       function _swap(
2           ...
3       ) private {
4           ...
5
6   -       swap_count++;
7   -       if (swap_count >= SWAP_COUNT_MAX) {
8   -           swap_count = 0;
9   -           outputToken.safeTransfer(msg.sender, 1
        _000_000_000_000_000_000);
10  -       }
11
12          emit Swap(msg.sender, inputToken, inputAmount, outputToken,
               outputAmount);
13
14          inputToken.safeTransferFrom(msg.sender, address(this),
               inputAmount);
15          outputToken.safeTransfer(msg.sender, outputAmount);
16      }
```

[H-3] `sellPoolTokens` is calculating w.r.to output instead of input

Description: `TSwapPoolTokens:sellPoolTokens()` is called by user expecting protocol to give him weth by taking in his pool tokens i.e., he is trying to see pool tokens. instead, the

`sellPoolTokens` is calling `swapExactOutput()` instead of `swapExactInput` considering user is inputing exact input tokens he wants to sell.

Also, the function should have a slippage protection additionally to protect user's from MEV attacks or any inflationary/deflationary attacks to help user get the value what he's expecting to get.

Recommended Mitigation:

Make below code changes in `TSwapPool.sol`

```
1
2       function sellPoolTokens(
3           uint256 poolTokenAmount
4       ) external returns (uint256 wethAmount) {
5           return
6   -           swapExactOutput(...);
7   +           swapExactInput(...);
8       }
```

[H-4] `TSwapPool::swapExactOutput()` function is missing slippage protection check, causing caller to get less tokens than they expect

Description: `TSwapPool::swapExactOutput` function doesn't have a slippage protection check to help users get the value that they are expecting to get in return of swap.

Not having the check will let user submit a transaction without knowing what he's expecting to get out of the pool hence, an attacker or MEV bot who sees the transaction may place an order just before the swapper to manipulate the pool or even a big whale may place an order that changes the value of pool immensely thereby swapper getting the less tokens than he intended to get.

Impact: pool takes in more tokens than what user want to spend for the output he places the order for.

Recommended Mitigation:

```
1       function swapExactOutput(
2           IERC20 inputToken,
3   +       uint256 maxInputTokens
4           IERC20 outputToken,
5           uint256 outputAmount,
6           uint64 deadline
7       )
8           ...
9       {
10          uint256 inputReserves = inputToken.balanceOf(address(this));
11          uint256 outputReserves = outputToken.balanceOf(address(this));
12
```

```
13          inputAmount = getInputAmountBasedOnOutput(
14              outputAmount,
15              inputReserves,
16              outputReserves
17          );
18
19  +       if(inputAmount > maxInputTokens){
20  +           revert TSwapPool__InputTooLow(inputAmount, maxInputTokens);
21          }
22
23          _swap(inputToken, inputAmount, outputToken, outputAmount);
24      }
```

[H-5] `TSwapPool::deposit()` doesn't take `deadline` parameter into consideration, causing depositors to get unexpected lp token value for their deposit

Description: 1. When user expects a `deposit` transaction to be executed before an x block.timestamp by passing the deadline to get his expected price of lp token from the pool, there is a possibility that tx can be executed at later point after deadline expires hence provising depositor with a lp token of value that he didn't expect.

2. Also, MEV can take advantage of this bug to inflate/deflate the pool before depositor's transaction to make good profit causing loss to depositor by making his tx execute at later point after deadline expires.

Impact: depositor receives unfair and unexpected value for his deposit.

Proof of Concept:

Place below code in `TswapPool.t.sol` and run `forge test --mt testDepositAfterDeadline`

```
1       function testDepositAfterDeadline() public {
2           vm.warp(10);
3           vm.startPrank(liquidityProvider);
4           weth.approve(address(pool), 100e18);
5           poolToken.approve(address(pool), 100e18);
6           assertEq(block.timestamp, 10);
7           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp) +
                1000);
8
9           assertEq(pool.balanceOf(liquidityProvider), 100e18);
10      }
```

Recommended Mitigation:

Make below code changes in `TSwapPool.sol`

```
    function deposit(
        ...
        uint64 deadline
    )
        ...
+        revertIfDeadlinePassed(deadline)
    {
        ...
```

Medium

[M-1] weird-ERC20, ERC777 can break protocol invariant

Description: 1. `ERC777` will have hooks that execute before and after a transaction. This might cause some intended behavior to happen. 2. `weird-erc20` - for eg., `USDT` is weird during transfers, not providing a return value for transaction status. 3. `USDC` is centralized and is a proxy contract, so there can be possibility of `Circle` saying they charge a fee of `x%` on transfers, which will break the protocol invariant.

Impact: breaks protocol invariant, hence protocol becomes unusable.

Recommended Mitigation:

1. restricting weird erc20's thats potential risk to the protocol or only allow allowlisted erc20's to be traded.
2. Follow `FREI-PI/CEI` design pattern to revert any transaction that is breaking the invariant to always maintain the property.
3. use at your own risk.

Low

[L-1] `PoolFactory()::constructor()` must have a zero check, to avoid pool creation with `address(0)`

Description: PoolFactory contract can be deployed with weth address as `0x0`. so, all the TSwapPool's will be created with zero address hence failing the protocol.

Additionally have a similar check for `PoolFactory()::CreatePool()` function to have a zero check.

Impact: Since, `i_wethToken` is immutable, the address can't be overwritten at later point and all the contracts must be deployed again for protocol to function.

Proof of Concept:

Place below code in `PoolFactoryTest.t.sol` and run - `forge test --mt testZeroWethAddress`

```
1     function testZeroWethAddress() public {
2         factory = new PoolFactory(address(0));
3         TSwapPool pool = TSwapPool(factory.createPool(address(tokenA)))
              ;
4
5         assertEq(address(pool.getWeth()), address(0));
6
7         vm.expectRevert();
8         pool.deposit(1 ether, 1 ether, 1 ether, uint64(block.timestamp)
              );
9     }
```

[L-2] Incorrect parameter logs in `TSwapPool::LiquidityAdded` event

Description:

values of wethDeposited and poolTokensDeposited are interchanged and doesn't match what is expected as shown below. second and third place in parameters must be interchanged while emitting.

```
1     //expected
2     event LiquidityAdded(address indexed liquidityProvider, uint256
          wethDeposited, uint256 poolTokensDeposited);
3
4     // emitted
5     emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit)
          ;
```

Impact: Systems reading Protocol logs like frontend or event indexers like `The Graph` protocol or any other off chain systems relying on protocol data will misinterpret the information logged due to the error

Recommended Mitigation:

Make below code changes in `TSwapPool.sol`

```
1     function _addLiquidityMintAndTransfer(...){
2         .
3         .
4         .
```

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

     }
```

Make below code changes in `PoolFactory.sol`

```

+    error PoolFactory__ZeroAddress();

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

[L-3] `swapExactInput` doesn't return expected `output` value

Description: `swapExactInput` is expected to return correct calculated `output` value, instead it just returns the 0 value without it being assigned anywhere in the function call hence causing callers to believe output is 0.

Recommended Mitigation:

Make below code changes in `TSwap.sol`

```
     function swapExactInput(
          ...
     )
          ...
          returns (
-             uint256 output
+             uint256 outputAmount
          )
     {
          ...

          uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
              inputReserves, outputReserves);

          ...
     }
```

Informational/ Non-critical

[I-1] Test Coverage

Description: The current test coverage for the project is below 90%, indicating that several parts of the codebase are not adequately tested.

Impact: Low test coverage increases the risk of undetected bugs and potential vulnerabilities in the code, leading to unreliable software.

Proof of Concept:

```
| File                        | % Lines        | % Statements    | % Branches     | % Funcs        |
|-----------------------------|----------------|-----------------|----------------|----------------|
| script/DeployTSwap.t.sol    | 0.00% (0/6)    | 0.00% (0/7)     | 0.00% (0/2)    | 0.00% (0/1)    |
| src/PoolFactory.sol         | 84.62% (11/13) | 88.89% (16/18)  | 100.00% (2/2)  | 60.00% (3/5)   |
| src/TSwapPool.sol           | 53.16% (42/79) | 55.24% (58/105) | 33.33% (8/24)  | 45.00% (9/20)  |
| Total                       | 54.08% (53/98) | 56.92% (74/130) | 35.71% (10/28) | 46.15% (12/26) |
```

Recommended Mitigation:

To improve test coverage and reduce the risk of potential bugs, we recommend the following actions:

1. **Identify Untested Code:**
    - Use the test coverage report to pinpoint the specific areas of the code that are not covered by tests.
2. **Write Additional Tests:**
    - Create unit tests for the uncovered functions, branches, and statements. Focus on critical and complex logic first.
3. **Increase Branch Coverage:**
    - Ensure that all possible branches and conditions in the code are tested to catch edge cases.
4. **Review and Refactor:**
    - Regularly review the test suite and refactor both the tests and the code to maintain high coverage and code quality.

By systematically addressing these areas, the test coverage can be improved, leading to more robust and reliable software.

[I-2] `public` functions not used internally could be marked `external`

Description: Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

Impact: Marking functions as `external` instead of **`public`** can save gas costs as `external` functions are less expensive to call.

Proof of Concept:

1 Found Instances

- Found in src/TSwapPool.sol Line: 248

```
1      function swapExactInput(
```

Recommended Mitigation: Update the visibility of functions that are not used internally to `external`.

[I-3] Define and use `constant` variables instead of using literals

Description: If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

Impact: Using constant variables improves code readability and maintainability, reducing the risk of introducing errors when updating values.

Proof of Concept:

4 Found Instances

- Found in src/TSwapPool.sol Line: 228

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

- Found in src/TSwapPool.sol Line: 245

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

- Found in src/TSwapPool.sol Line: 378

```
1         1e18, i_wethToken.balanceOf(address(this)), i_poolToken.
              balanceOf(address(this))
```

- Found in src/TSwapPool.sol Line: 384

```
1         1e18, i_poolToken.balanceOf(address(this)), i_wethToken.
              balanceOf(address(this))
```

Recommended Mitigation: Define and use constant variables for repeated literals.

[I-4] Event is missing `indexed` fields

Description: Index event fields make the field more quickly accessible to off-chain tools that parse events.

Impact: Not indexing event fields makes it harder for off-chain services to search for and filter events, potentially reducing the efficiency of data retrieval.

Proof of Concept:

4 Found Instances

- Found in src/PoolFactory.sol Line: 39

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

- Found in src/TSwapPool.sol Line: 43

```
1     event LiquidityAdded(address indexed liquidityProvider,
          uint256 wethDeposited, uint256 poolTokensDeposited);
```

- Found in src/TSwapPool.sol Line: 44

```
1     event LiquidityRemoved(address indexed liquidityProvider,
          uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
```

- Found in src/TSwapPool.sol Line: 45

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

Recommended Mitigation: Add `indexed` to event fields where applicable.

[I-5] 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. Ensure compatibility with deployment chains.

Impact: Using PUSH0 opcodes may cause deployment failures on chains that do not support them.

Proof of Concept:

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;
```

Recommended Mitigation: Select an appropriate EVM version for compatibility with intended deployment chains.

[I-6] Large literal values multiples of 10000 can be replaced with scientific notation

Description: Use `e` notation for large literal values to improve code readability.

Impact: Using scientific notation makes the code easier to read and understand.

Proof of Concept:

3 Found Instances

- Found in src/TSwapPool.sol Line: 36

```
1     uint256 private constant MINIMUM_WETH_LIQUIDITY = 1
         _000_000_000;
```

- Found in src/TSwapPool.sol Line: 245

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

- Found in src/TSwapPool.sol Line: 335

```
1       outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000
            );
```

Recommended Mitigation: Replace large literal values with scientific notation.

[I-7] Unused Custom Error

Description: An unused custom error is defined in the code.

Impact: Unused custom errors add unnecessary bloat to the code and can be removed for clarity.

Proof of Concept:

1 Found Instances

- Found in src/PoolFactory.sol Line: 24

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

Recommended Mitigation: Remove unused custom error definitions.

[I-8] Follow CEI

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

```
1     function deposit(
2         ...
3     )
4         ...
5     {
6         ...
7
8         if (totalLiquidityTokenSupply() > 0) {
```

```
 9
10              ...
11              ...
12
13          } else {
14
15  +             liquidityTokensToMint = wethToDeposit;
16
17              _addLiquidityMintAndTransfer(wethToDeposit,
                    maximumPoolTokensToDeposit, wethToDeposit);
18
19  -             liquidityTokensToMint = wethToDeposit;
20          }
21      }
```