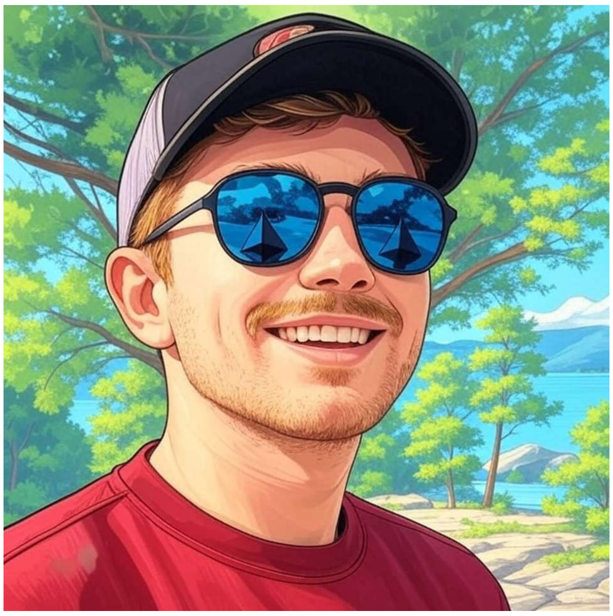

TSwap Protocol Audit Report

Version 1.0

Mustby.eth

July 16, 2025

TSwap Audit Report

Mustby.eth

July 16, 2025

Prepared by: Mustby.eth Lead Auditors: - Mustby.eth

Table of Contents

- Medium
  * [M-1] TITLE `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline.
  * [M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant.
- Low
  * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emit incorrect information.
  * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given.
- Informationals
  * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed.
  * [I-2] `PoolFactory::constructor` is missing a zero address check.
  * [I-3] Incorrect naming convention in `PoolFactory::createPool`. It should use `.symbol()` instead of `.name()`.
  * [I-4] Event is missing `indexed` fields
  * [I-5] Missing zero address check in `TSwapPool::constructor`.
  * [I-6] We don't need to emit `MINIMUM_WETH_LIQUIDITY` because it is a constant
  * [I-7] liquityTokensToMint does not follow CEI in `TSwapPool::deposit`.
  * [I-8] Magic numbers should be initialized to improve code readability in `TSwapPool::getOutputAmountBasedOnInput`.
  * [I-9] Magic numbers should be initialized to improve code readability in `TSwapPool::getInputAmountBasedOnOutput`.
  * [I-10] Add natspec in `TSwapPool::getInputAmountBasedOnOutput` to improve code readability.
  * [I-11] Add natspec in `TSwapPool::swapExactOutput` to improve code readability.
- Gas Findings
  * [G-1] poolTokenReserves in `TSwapPool::deposit` is not used.
  * [G-2] Change visibility in `TSwapPool::swapExactInput` to save gas.
  * [G-3] Change visibility in `TSwapPool::swapExactOutput` to save gas.
  * [G-4] Change visibility in `TSwapPool::totalLiquidityTokenSupply` to save 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 Mustby.eth 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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
  - Any ERC20 token

Scope

- In Scope:

```
./src/
#-- PoolFactory.sol
#-- TSwapPool.sol
```

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

The audit went very well. We found 23 vulnerabilities that must be addressed in order to ensure a smoothly running protocol.

Issues found

Severity	Number of Issues Found
High	4
Medium	2
Low	2
Info	11
Gas	4
Total	23

Findings

High

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

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

Impact: Protocol takes more fees than expected from users.

Proof of Concept: Write PoC showing how much we expect a fee to be versus how much it ACTUALLY is…

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10_000) / ((outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1_000) / ((outputReserves - outputAmount) * 997);
    }
```

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

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

Impact: If market conditions change before the transaction processes, the user could get a much worse swap.

Proof of Concept: Write my PoC! Prove the slippage. 1. The price of WETH right now is 1_000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = any 3. The function does not offer a max input 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 the 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-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens.

Description: The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and receive 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: Practice my Poc

Recommended Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` 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))
    }
```

Additionality, it might be wise to add a deadline to the function, as there is currently no deadline. (MEV will be handled later)

[H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the invariant of `x * y = k`.

Description: The protocol follows a strict invariant of `x * y = k`. Where: - `x`: The balance of the pook 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 the extra incentive in the 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 of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol.

Most simply put, the protocol's core invariant is broken.

Proof of Concept: 1. A user swaps 10 times, and collects the extra incentive of `1_000_000_000_000_000_000` tokens. 2. That user just continues to swap until all the protocol funds are drained.

Proof Of Code

Place the following into `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; // 0.1 WETH

        // swap
        vm.startPrank(user); // this means we are impersonating the swapper
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 10e18); // mint some pool tokens to the user for swapping
        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));
        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(); // this means we are stopping the impersonation of the swapper

        // actual output

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

Recommended Mitigation: Remove the extra incentive mechanism. If you 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 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] TITLE `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline.

Description: The `deposit` functions 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, operations 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,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
```

[M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant.

Description: The `_swap` function does not account for rebase, fee-on-transfer, or ERC777 tokens, which can disrupt the expected input/output amounts due to token balance changes during transfers, breaking the protocol's invariant of precise token swaps.

Impact: Users may receive incorrect output amounts, leading to potential loss of funds or manipulation of the swap mechanism, undermining the protocol's reliability.

Proof of Concept: 1. A fee-on-transfer token deducts a fee during `inputToken.safeTransferFrom`, causing the contract to receive less than `inputAmount`, but the function assumes the full amount is received. 2. A rebase token may alter balances unexpectedly, causing discrepancies in `inputAmount` or `outputAmount`. 3. An ERC777 token's callback (e.g., `tokensReceived`) could reenter the contract, potentially manipulating `swap_count` or triggering unintended transfers.

Recommended Mitigation: 1. Check the contract's balance before and after `safeTransferFrom` to ensure the exact `inputAmount` is received. 2. Disallow rebase and ERC777 tokens by maintaining a token allowlist or checking for unexpected balance changes. 3. Use a reentrancy guard to prevent ERC777 callback exploits.

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emit incorrect information.

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 `wethToDeposit` value should go second.

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

Recommended Mitigation:

```
-     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` function is expected to return the actual amount of tokens bought by the caller. However, while it declares the named return value `output` it is never assigned a value, nor uses a explicit return statement.

Impact: The return value will always be zero, giving incorrect information to the caller.

Proof of Concept: PoC on the returned zeros. Show that no matter what we swap, we will always get zero.

Recommended Mitigation:

```
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-       uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);
+       output = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);

-        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);
    }
```

Informationals

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

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

[I-2] `PoolFactory::constructor` is missing a zero address check.

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

[I-3] Incorrect naming convention in `PoolFactory::createPool`. It should use `.symbol()` instead of `.name()`.

```
    function createPool(address tokenAddress) external returns (address) {
```

```diff
        if (s_pools[tokenAddress] != address(0)) {
            revert PoolFactory__PoolAlreadyExists(tokenAddress);
        }
        string memory liquidityTokenName = string.concat("T-Swap ",
           IERC20(tokenAddress).name());
-        string memory liquidityTokenSymbol = string.concat("ts", IERC20
(tokenAddress).name());
+        string memory liquidityTokenSymbol = string.concat("ts", IERC20
(tokenAddress).symbol());
        TSwapPool tPool = new TSwapPool(tokenAddress, i_wethToken,
           liquidityTokenName, liquidityTokenSymbol);
        s_pools[tokenAddress] = address(tPool);
        s_tokens[address(tPool)] = tokenAddress;
        emit PoolCreated(tokenAddress, address(tPool));
        return address(tPool);
    }
```

[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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 43

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

- Found in src/TSwapPool.sol Line: 44

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

- Found in src/TSwapPool.sol Line: 45

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

[I-5] Missing zero address check in `TSwapPool::constructor`.

```
    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] We don't need to emit `MINIMUM_WETH_LIQUIDITY` because it is a constant

Description: This value will never be changed so there is no point in emitting it.

Recommended Mitigation:

In `TSwapPool`:

```
-       error TSwapPool__WethDepositAmountTooLow(uint256
    minimumWethDeposit, uint256 wethToDeposit);
+       error TSwapPool__WethDepositAmountTooLow(uint256 wethToDeposit)
    ;
```

In the `TSwapPool::deposit` function:

```
    if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
-       revert TSwapPool__WethDepositAmountTooLow(
    MINIMUM_WETH_LIQUIDITY, wethToDeposit);
+       revert TSwapPool__WethDepositAmountTooLow(wethToDeposit);
        }
```

[I-7] liquityTokensToMint does not follow CEI in `TSwapPool::deposit`.

Recommended Mitigation: Move the liquidityTokensToMint line above _addLiquidityMintAndTransfer.

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

[I-8] Magic numbers should be initialized to improve code readability in `TSwapPool::getOutputAmountBasedOnInput`.

Recommended Mitigation:

Initialize values up top:

```
+   uint256 private constant PRECISION_FACTOR = 1000;
+   uint256 private constant FEE_FACTOR = 997;
```

Fix the values in the code:

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

[I-9] Magic numbers should be initialized to improve code readability in `TSwapPool::getInputAmountBasedOnOutput`.

Recommended Mitigation:

We alredy initialized these values up top in [I-8].

```
+   uint256 private constant PRECISION_FACTOR = 1000;
+   uint256 private constant FEE_FACTOR = 997;
```

Fix the values in the code:

```
    {
-        return ((inputReserves * outputAmount) * 10000) / ((
   outputReserves - outputAmount) * 997);
+        return ((inputReserves * outputAmount) * PRECISION_FACTOR) /
   ((outputReserves - outputAmount) * FEE_FACTOR);
    }
```

[I-10] Add natspec in `TSwapPool::getInputAmountBasedOnOutput` to improve code readability.

Recommended Mitigation: Add this above the `TSwapPool::getInputAmountBasedOnOutput` function:

```
    /// @notice Calculates the required input amount of one token to
        receive a specified output amount of another token, accounting
        for pool reserves and fees.
    /// @param outputAmount The amount of output tokens desired.
    /// @param inputReserves The current reserves of the input token in
         the pool.
    /// @param outputReserves The current reserves of the output token
        in the pool.
    /// @return inputAmount The calculated amount of input tokens
        required to receive the specified output amount.
```

[I-11] Add natspec in `TSwapPool::swapExactOutput` to improve code readability.

Recommended Mitigation: Add this above the `TSwapPool::swapExactOutput` function:

```
+    /// @param deadline The deadline for the transaction to be
    completed by
```

Gas Findings

[G-1] poolTokenReserves in `TSwapPool::deposit` is not used.

Recommended Mitigation: Remove this line to save gas.

```
    if (totalLiquidityTokenSupply() > 0) {
            uint256 wethReserves = i_wethToken.balanceOf(address(this))
                ;
-             uint256 poolTokenReserves = i_poolToken.balanceOf(address(
    this));
```

[G-2] Change visibility in `TSwapPool::swapExactInput` to save gas.

Description: This function is not called internally so we should make it external to save gas.

Recommended Mitigation:

```
    function swapExactInput(...)
-       public (...)
+       external (...)
```

[G-3] Change visibility in `TSwapPool::swapExactOutput` to save gas.

Description: This function is not called internally so we should make it external to save gas.

Recommended Mitigation:

```
    function swapExactOutput(...)
-       public (...)
+       external (...)
```

[G-4] Change visibility in `TSwapPool::totalLiquidityTokenSupply` to save gas.

Description: This function is not called internally so we should make it external to save gas.

Recommended Mitigation:

```
    function totalLiquidityTokenSupply()
-       public view returns (uint256) {...}
+       external view returns (uint256) {...}
```