# TSwap Audit Report

Prepared by: Lhoussain Ait Aissa Lead Security Researcher:

# Table of contents

▶ Details

See table


- TSwap Audit Report
- Table of contents
- About Lhoussaine Ait Aisaa
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
- Protocol Summary
  - Roles
- Executive Summary
  - Issues found
- Findings
- PoolFactory
  - Informations
    - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error doesn't use it and should remove it
    - [I-2] In `PoolFactory::Constructor()` loaking zero check addresses
    - [I-3] In `PoolFactory::createPool()` using `.name()` instead of `.symbol()`
- TSwapPool
  - High
    - [H-1] The `sellPoolTokens` function miscalculates amount of tokens bought
    - [H-2] Protocol may take too many tokens from users during swap, resulting is lost fee
    - [H-3] Additional swap incentive breaks protocol invariant
  - Meduim

        - [M-1] In `TSwapPool::deposit` function missing `deadline` check causing the transaction to complite even after the `deadline`
        - [M-2] Lack of slippage protection in `swapExactOutput` function
        - [M-3] In `TSwapPool::getInputAmountBasedOnOutput` function existe an arithmetic isseu ,causing a divice by zero
    - Low
        - [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
        - [L-2] Swapping function returns default value
    - Informations
        - [I-1] Event is missing `indexed` fields
        - [I-2] In `TSwapPool::Constructor()` loaking zero check addresses


# About Lhoussaine Ait Aisaa

I am a dedicated Smart Contract Developer and Security Researcher with 2 years of hands-on experience in writing efficient and secure Solidity code. I possess strong expertise in auditing smart contracts to ensure robust security and functionality. I am seeking to leverage my technical skills and attention to detail to contribute to innovative blockchain projects within a dynamic team.

# Disclaimer

The Lhoussaine Ait Aisaa 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 |

# Audit Details

**The findings described in this document correspond the following commit hash:**

```
1ec3c30253423eb4199827f59cf564cc575b46db
```

## Scope

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

# Protocol Summary

TSWAP is an constant-product AMM that allows users permissionlessly trade WETH and any other ERC20 token set during deployment. Users can trade without restrictions, just paying a tiny fee in each swapping operation. Fees are earned by liquidity providers, who can deposit and withdraw liquidity at any time.

## Roles

- Liquidity Provider: An account who deposits assets into the pool to earn trading fees.
- User: An account who swaps tokens.

# Executive Summary

## Issues found

| Severity | Number of issues found |
| --- | --- |
| High | 3 |
| Medium | 3 |
| Low | 2 |
| Info | 3 + 2 |
| Gas | 0 |
| Total | 13 |

# Findings

# PoolFactory

## Informations

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error doesn't use it and should remove it

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

[I-2] In `PoolFactory::Constructor()` loaking zero check addresses

```
    constructor(address wethToken) {

+        if (wethToken != address(0)){

        i_wethToken = wethToken;

+        }
    }
```

[I-3] In `PoolFactory::createPool()` using `.name()` instead of `.symbol()`

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

# TSwapPool

## High

### [H-1] The `sellPoolTokens` function miscalculates amount of tokens bought

The `sellPoolTokens` is intended to allow users easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell using 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` is the one that should be called. Because users specify the exact amount of input tokens - not output tokens.

Consider changing the implementation to use the `swapExactInput` function. Note that this would also require to change the `sellPoolTokens` function to accept a new parameter (e.g., `minWethToReceive`) to be passed down to `swapExactInput`.

```
    function sellPoolTokens(
        uint256 poolTokenAmount
+       uint256 minWethToReceive
    ) external returns (uint256 wethAmount) {
-       return swapExactOutput(
+       return swapExactInput(
```

```
            i_poolToken,
            poolTokenAmount,
            WETH_TOKEN,
+           minWethToReceive,
            uint64(block.timestamp)
        );
    }
```

### [H-2] Protocol may take too many tokens from users during swap, resulting is lost fee

The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of output tokens. However, the function currently miscalculates the resulting amount. When calculating the fee, it scales the amount by 10000 instead of 1000.

```
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);
    }
```

As a result, users swapping tokens via the `swapExactOutput` function will pay far more tokens than expected for their trades. This becomes particularly risky for users that provide infinite allowance to the `TSwapPool` contract. Moreover, note that the issue is worsened by the fact that the `swapExactOutput` function does not allow users to specify a maximum of input tokens, as is described in another issue in this report.

It's worth noting that the tokens paid by users are not lost, but rather can be swiftly taken by liquidity providers. Therefore, this contract could be used to trick users, have them swap their funds at unfavorable rates and finally rug pull all liquidity from the pool.

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 WETH 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 tokens, and
    now only has less than 1.
        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);
    }
```

## [H-3] Additional swap incentive breaks protocol invariant

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

# Meduim

## [M-1] In `TSwapPool::deposit` function missing `deadline` check causing the transaction to complite even after the `deadline`

**Description:** The `deposit` function accepts a `deadline` parameter, which according to 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 for the caller.

**Recommended Mitigation:** Consider making the following change to the `deposit` function:

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

## [M-2] Lack of slippage protection in `swapExactOutput` function

The `swapExactOutput` function does not include any sort of slippage protection to protect user funds that swap tokens in the pool. Similar to what is done in the `swapExactInput` function, it should include a parameter (e.g., `maxInputAmount`) that allows callers to specify the maximum amount of tokens they're willing to pay in their trades.

```
function swapExactOutput(
    IERC20 inputToken,
+   uint256 maxInputAmount
    IERC20 outputToken,
    uint256 outputAmount,
    uint64 deadline
)
    public
    revertIfZero(outputAmount)
    revertIfDeadlinePassed(deadline)
    returns (uint256 inputAmount)
{
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));

    inputAmount = getInputAmountBasedOnOutput(outputAmount, inputReserves,
outputReserves);
```

```
+   if (inputAmount > maxInputAmount) {
+       revert TSwapPool__OutputTooHigh(inputAmount, maxInputAmount);
+   }

    _swap(
        inputToken,
        inputAmount,
        outputToken,
        outputAmount
    );
}
```

### [M-3] In `TSwapPool::getInputAmountBasedOnOutput` function existe an arithmetic isseu ,causing a divice by zero

**Description:** When `outputReserves` & `outputAmoun` in `TSwapPool::getInputAmountBasedOnOutput` are the same , its substraction equal zero , and that causing a divice by zero :

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {

        return
            ((inputReserves * outputAmount) * 10000) /
@>            ((outputReserves - outputAmount) * 997);
    }
```

**Impact:** Divice by zero

**Proof of Concept:**

Place this in TSwapPool.t.sol file :

```
        function testRevertIfOutputsArethesame() public {

            vm.startPrank(user);
            weth.approve(address(pool), 100e18);
            pool.approve(address(pool), 100e18);
```

```
            uint256 OutputAmout = weth.balanceOf(user);
            uint256 InputAmout = pool.balanceOf(user);

            vm.expectRevert();
            tswap.getInputAmountBasedOnOutput(OutputAmout,InputAmout,OutputAmout);

        }
```

**Recommended Mitigation:** Add a custum error :

```
+    error OutputsAreThesame();
.
.
.

    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {

+        if(outputReserves == outputAmout){

+            revert OutputsAreThesame();

+        }

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

## Low

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

**Description:** When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` function , it logs value in incorrect order. the

`poolTokensToDeposit` value should go in the third parameter position , whereas the `wethToDeposit` value should go secound .

**Impact:** Event emission is incorrect , loading to on-chain function potentially malfunctioning

**Recommended Mitigation:**

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

### [L-2] Swapping function returns default value

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 never assigns a value to it, nor uses an explicit `return` statement.

As a result, the function will always return zero. Consider modifying the function so that it always return the correct amount of tokens bought by the caller.

## Informations

### [I-1] 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: 1307:61:31
- Found in src/TSwapPool.sol: 1897:108:10
- Found in src/TSwapPool.sol: 2010:110:10
- Found in src/TSwapPool.sol: 2125:116:10

### [I-2] In `TSwapPool::Constructor()` loaking zero check addresses

```
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {

+        if(poolToken == address(0) || wethTToken == address(0)){
+            revert();
+        }

        i_wethToken = IERC20(wethToken);
```

```
        i_poolToken = IERC20(poolToken);

    }
```