TSwap Security Review

Prepared by: Fabriziogianni7
Lead Auditors:

- Fabriziogianni7

Table of Contents

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

Fabriziogianni7 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

Files Summary

Extension	**Value**
.sol Files	2

Extension	Value
Total nSLOC	262

Files Details

Filepath	nSLOC
src/PoolFactory.sol	35
src/TSwapPool.sol	227
Total	**262**

Issue Summary

Category	No. of Issues
Critical	0
High	5
Medium	2
Low	1
Info	5

Findings

High

[H-1] Sending a "bonus fee" after `TSwapPool::SWAP_COUNT_MAX` is reached is breaking x * y = k invariant

Description:
`TSwapPool::_swap` function is sending an arbitrary amount of Tokens to whom is swapping the the count of swap reaches
`TSwapPool::SWAP_COUNT_MAX`. This causes the protocol to lose liquidity

Impact:
Critical: since `TSwapPool::_swap` function is the core of the protocol, the likelihood of this bug happening is very high as well as the damage that can cause
Proof Of Concept:
Recommended Mitigation:
remove this dynamic or use an approach more similar on how the fees are accrued

[H-2] `TSwapPool::deposit` function does not check the deadline and makes the transaction subject to long-pending transactions and wild swings in price, making deposit conditions unfavorable for the lp.

Description:
`TSwapPool::deposit` has a parameter `deadline` but is not using it.

Impact:
High: `TSwapPool::deposit` function the entrypoint to liquidity providers, they can deposit large amount of tokens and be subject to change of prices and MEV. Likelyhood is also high since this always happen every time users call this method.

Proof Of Concept:
`deadline` parameter is never used:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline // @audit-info unused parameters
    )
        external
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {...}
```

Recommended Mitigation:
change the function adding the correct modifier

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline // @audit-info unused parameters
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {...}
```

[H-4] `TSwapPool::getInputAmountBasedOnOutput` calculate fees wrongly, uses 10k/997 instead of 1k/997

Description:
`TSwapPool::getInputAmountBasedOnOutput` uses a magic number to calculate the fee. it scales the amount of fee to 10_000 instead of 1_000.

Impact:
Likelyhood: High. It happens every time this method is invoked
Impact: High. the fees for the inputAmount are going to be very high.

Proof Of Concept:
here the test:

```
function testFeesMiscalculation() public {
        //deposit in pool

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

        uint256 outputAmount = 2e18;
        uint256 inputReserves = poolToken.balanceOf(address(pool));
        uint256 outputReserves = weth.balanceOf(address(pool));
        uint256 inputAmountFor2Weth =
pool.getInputAmountBasedOnOutput(outputAmount, inputReserves,
outputReserves);

        uint256 shouldBe = ((inputReserves * outputAmount) * 1_000) /
((outputReserves - outputAmount) * 997);

        console.log("shouldBe %", shouldBe);
        console.log("inputAmountFor2Weth %", inputAmountFor2Weth);
        assertGt(inputAmountFor2Weth, shouldBe);
    }
```

the test above logs

```
shouldBe % 2046957198124987206
inputAmountFor2Weth % 20469571981249872065
```

Recommended Mitigation:

- use magic numbers
- change

```
+ return ((inputReserves * outputAmount) * 1_000) / ((outputReserves -
outputAmount) * 997);
- return ((inputReserves * outputAmount) * 10000) / ((outputReserves -
outputAmount) * 997);
```

[H-5] Slippage vulnerability in `TSwapPool::swapExactOutput` due to missing maxInputTokenAmount

Description:
`TSwapPool::swapExactOutput` function does not include any anti-slippage measure. the function should specify a maximum amount of token that the user allows to input.

Impact:
Likelyhood: Medium-High. It happens every time this method is invoked.
Impact: High. If market conditions change before the execution of the function is completed, the user can get bad swapping conditions.

Proof Of Concept:

1. The price of 1 WETH before swap is 1,000 USDC
2. User inputs a swapExactOutput looking for 1 WETH
    - inputToken = USDC
    - outputToken = WETH
    - outputAmount = 1
    - deadline = whatever
    - The function does not offer a maxInput amount
3. As the transaction is pending in the mempool market condition change:
    - 1 WETH is now 10,000 USDC. 10x more than the user expected
4. The transaction completes, but the user sent the protocol 10,000 USDC instead of the expected 1,000 USDC

Recommended Mitigation:
Include a `maxInputTokenAmount` parameter in the method

```
function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
+       uint256 maxInputTokenAmount
    )
```

[H-6] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens

Description: In `TSwapPool::sellPoolTokens` `swapExactOutput` is used to sell pool tokens in exchange of WETH.
The argument `poolTokenAmount` in `sellPoolTokens` is used as 3rd input in `swapExactOutput`. `swapExactOutput` accept `outputAmount`as a 3rd input. This should be the amount of Weth.

Using `swapExactOutput` is non logical, suggesting using `swapExactInput`.

Impact: Likelyhood: High. It happens every time this method is invoked.
Impact: High. Users will swap the wrong amount of tokens, which is a `severe disruption of protcol functionality`.

Recommended Mitigation:

- use `swapExactInput` instead of `swapExactOutput`.

Medium

[M-1] 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: 157

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

[M-2] `TSwapPool::_addLiquidityMintAndTransfer` emit `TSwapPool::LiquidityAdded` event with wrong parameters returning bad informations to external services listening to this contract

Low

[L-1] `TSwapPool::swapExactInput` has an unused return value, causing the function to always return 0, wrong data.

Description:
This is the function:

```
function swapExactInput(
       IERC20 inputToken,
       uint256 inputAmount,
       IERC20 outputToken,
       uint256 minOutputAmount,
       uint64 deadline
   )
       public
       revertIfZero(inputAmount)
       revertIfDeadlinePassed(deadline)
       returns (
           // unused variable
           uint256 output
       )
```

Recommended Mitigation:
remove the return value

```
function swapExactInput(
       IERC20 inputToken,
```

```
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
-       returns (
-           // unused variable
-           uint256 output
-       )
```

Informational - Gas

[I-1] `TSwapPool::deposit` has an unused variable ``TSwapPool::sellPoolTokens` remove

[I-2] `TSwapPool::_addLiquidityMintAndTransfer` natspec not clear: `This is a sensitive function, and should only be called by addLiquidity` should be: `This is a sensitive function, and should only be called by deposit`

[I-3] `TSwapPool::getOutputAmountBasedOnInput`, `TSwapPool::getInputAmountBasedOnOutput`, uses magic numbers.

[I-4] Missing natspec on fundamental methods `getOutputAmountBasedOnInput`, `getInputAmountBasedOnOutput`, `swapExactInput`, `getPoolTokensToDepositBasedOnWeth`

[I-5] `PoolFactory` has an unused error `PoolFactory__PoolDoesNotExist` remove