TSwap Protocol Audit Report

Version 1.0

plairfx.xyz

March 21, 2024

TSwap Audit

Plairfx

21 March 2024

Prepared by: plairfx Lead Auditors: - plairfx

Table of Contents

- Medium
  - [M-01] in `TSwapPool.sol::Deposit uint64 deadline` is never used causing transactions to be completed after the deadline!
  - [M-02] In the `PoolFactory:CreatePool` function, when generating a liquidity pool token with a symbol, it mistakenly utilizes `.name` instead of `.symbol`.
  - [M-03] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant.
- Lows
  - [L-01] in `TSwapPool:SwapExactInput` uint256 `output` returned value is incorrect, it does not return anything.
  - [L-02] `Event:LiquidityAdded` is emittted backwards, which causes the wrong information to be emitted as an event.
  - [L-03] If an event has more than 3 events,it should be indexed, this will allow data to be more avaliable to offchain-tools.
  - [L-04] In `TSwapPool.sol:SwapExactInput` has no natspec.
- Informationals
  - [I-01] In `TSwapPool.sol:Deposit uint256 poolTokenReserves` is never used in the function again.
- **Proof of Concept:**
  - [I-02] in `PoolFactory` error `error PoolFactory__PoolDoesNotExist(address tokenAddress);` is not being used.

Protocol Summary

Protocol does X, Y, Z

Disclaimer

Plairfx 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 plairfx 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
Likelihood	High	H	H/M	M
	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
```

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

This was a protocol that helped me get into the DeFi world especially this fork of uniswap V1, gave me an overview of the many issues that could arise from the weird erc20's tokens, (OHM) for example.

Much was learnt, while writing the POC's, and exploring the fuzz tests. ## Issues found

Severity	Number of issues found
High	4
Medium	3
Low	3
Info	2
Gas	0
Total	12

Findings

High

[H-01] The absence of implementation for `MaxinputAmount` in `SwapExactOutPut` introduces significant issues and lacks essential safety checks. Missing Slippage Protection.

Description: The function `SwapExactOutPut` has no slippage protection, essentialy if this transaction

Impact: This oversight enables users to proceed with transactions exceeding the designated inputAmount. The absence of a check against the maxInputAmount permits trades or swaps beyond protocol limits, potentially leading to detrimental outcomes.

Proof of Concept:

```
    function testIfThereIsSlippage() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), type(uint256).max);
        poolToken.approve(address(pool), type(uint256).max);
        pool.deposit(100e18, 0, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        // Start prank for user
        vm.startPrank(user);
        poolToken.mint(user, 80e18);
        poolToken.approve(address(pool), 80e18);


```

```
16          pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
                timestamp));
17          vm.stopPrank();
18
19          console.log("weth.balanceOf(user)", poolToken.balanceOf(user));
20          console.log("expected", 1 ether);
```

we expect 1 ether, and we get less than 1 ether, no slippage protection means the users will not be protected.

Recommended Mitigation:

```
1  + error TSwapPool__InputTooHigh(uint256 inputAmount,  uint256
       MaxInputAmount);
2
3
4   function swapExactOutput(
5          IERC20 inputToken,
6          IERC20 outputToken,
7          uint256 outputAmount,
8  +       uint256 MaxinputAmount,
9          uint64 deadline
10     )
11
12         public
13         revertIfZero(outputAmount)
14         revertIfDeadlinePassed(deadline)
15         returns (uint256 inputAmount)
16     {
17         uint256 inputReserves = inputToken.balanceOf(address(this));
18         uint256 outputReserves = outputToken.balanceOf(address(this));
19
20 +           if (inputAmount > maxInputAmount) {
21 +           revert TSwapPool__OutputTooLow(inputAmount, MaxINputAmount)
       ; // Of course make a  new error message.
22         }
23
24         inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
25
26         _swap(inputToken, inputAmount, outputToken, outputAmount);
27     }
```

[H-02] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutPut` causes a 91.3% FEE to the user! and resulting in lost fees and user tokens.

Description: Inputting an additional zero, such as typing `10000` instead of `1000`, triggers the protocol to impose an exorbitant 91.7% fee. This catastrophic error can lead to the collapse of the entire protocol

and result in substantial losses for users.

Impact: This will cause the protocol to have a 91.3% FEE instead of a 0.3% essentialy taking all the users money and leaving them with 8.7%!! instead of 99.7% of their money!

Proof of Concept:

javascript“‘ function testgetInputAmountBasedOnOutput() public { uint256 outputAmount = 100; uint256 inputReserves = 1000; uint256 outputReserves = 1000; uint256 input = pool.getInputAmountBasedOnOutput(ou inputReserves, outputReserves); console.log(“input”, input);

```
    uint256 normalresult = ((inputReserves * outputAmount) * 1000) / ((
        outputReserves - outputAmount) * 997);

    console.log("normalresult", normalresult);

    assertEq(normalresult, input);
}
```

}“‘

The normal result should be 111, put the mistake makes it 1114..

Recommended Mitigation: First of all the goal is to use magic numbers, to overcome this issue at all times, This essentially takes care of two issues at ones, magic numbers and typo while typing

You can do that by setting it as

```
uint256 private constant WHOLE_RATE = 1000;
uint256 private constant FEE_RATE = 997;

- ((inputReserves * outputAmount) * 10000) / ((outputReserves -
    outputAmount) * 997);

+ uint256 private constant WHOLE_RATE = 1000;
+ uint256 private constant FEE_RATE = 997;
+  ((inputReserves * outputAmount) * WHOLE_RATE) / ((outputReserves -
    outputAmount) * FEE_RATE);
```

This will solve the issues.

[H-03] Swap breaks the invariant at the 10th swap.

Description: When swapping the protocol will give out an extra token as an extra incentive! Which breaks the invariant!

```
  /**
     * @notice Swaps a given amount of input for a given amount of
        output tokens.
```

```
     * @dev Every 10 swaps, we give the caller an extra token as an
         extra incentive to keep trading on T-Swap.
     * @param inputToken ERC20 token to pull from caller
     * @param inputAmount Amount of tokens to pull from caller
     * @param outputToken ERC20 token to send to caller
     * @param outputAmount Amount of tokens to send to caller
     */
    function _swap(IERC20 inputToken, uint256 inputAmount, IERC20
        outputToken, uint256 outputAmount) private {
        if (_isUnknown(inputToken) || _isUnknown(outputToken) ||
            inputToken == outputToken) {
            revert TSwapPool__InvalidToken();
        }

        swap_count++;
        if (swap_count >= SWAP_COUNT_MAX) {
            swap_count = 0;
            outputToken.safeTransfer(msg.sender, 1
                _000_000_000_000_000_000);
        }
        emit Swap(msg.sender, inputToken, inputAmount, outputToken,
            outputAmount);

        inputToken.safeTransferFrom(msg.sender, address(this),
            inputAmount);
        outputToken.safeTransfer(msg.sender, outputAmount);
    }
```

Impact: This will break the invariant of the pool balances staying the same, as the protocol is giving an extra incentive!

Proof of Concept:

```
function testDepositSwapFee10() public {
        for (uint256 i = 0; i < 11; i++) {
            // Start prank for liquidityProvider
            vm.startPrank(liquidityProvider);
            weth.approve(address(pool), 100e18);
            poolToken.approve(address(pool), 100e18);
            pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp
                ));
            vm.stopPrank();

            // Start prank for user
            vm.startPrank(user);
            console.log("weth.balanceOf(user)1", weth.balanceOf(user));
            poolToken.approve(address(pool), 10e18);

            uint256 BeforeBalance = weth.balanceOf(user);
            pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block
                .timestamp));
```

```
17
18              console.log("weth.balanceOf(user)", weth.balanceOf(user));
19
20              assertEq(weth.balanceOf(user), BeforeBalance);
21              vm.stopPrank();
22          }
23
24      }
```

```
1  **Recommended Mitigation:**
2  ```diff
3    /**
4       * @notice Swaps a given amount of input for a given amount of
            output tokens.
5       * @dev Every 10 swaps, we give the caller an extra token as an
            extra incentive to keep trading on T-Swap.
6       * @param inputToken ERC20 token to pull from caller
7       * @param inputAmount Amount of tokens to pull from caller
8       * @param outputToken ERC20 token to send to caller
9       * @param outputAmount Amount of tokens to send to caller
10      */
11      function _swap(IERC20 inputToken, uint256 inputAmount, IERC20
           outputToken, uint256 outputAmount) private {
12          if (_isUnknown(inputToken) || _isUnknown(outputToken) ||
               inputToken == outputToken) {
13              revert TSwapPool__InvalidToken();
14          }
15
16  -       swap_count++;
17  -       if (swap_count >= SWAP_COUNT_MAX) {
18  -           swap_count = 0;
19  -           outputToken.safeTransfer(msg.sender, 1
        _000_000_000_000_000_000);
20  -       }
21          emit Swap(msg.sender, inputToken, inputAmount, outputToken,
               outputAmount);
22
23          inputToken.safeTransferFrom(msg.sender, address(this),
               inputAmount);
24          outputToken.safeTransfer(msg.sender, outputAmount);
25      }
```

[H-04] In `Tswappool.sol:sellPoolTokens` mismatchig the input and output causing the user not to receive the correct amount of tokens

Description: The `sellPoolTokens` function is intended to enable users to exchange their pool tokens for WETH. However, it currently calls the `swapExactOutput` function instead of `swapExactInput`, which is the correct function to use for selling tokens.

In the current implementation, users specify the amount of WETH they wish to receive, but ideally, they should input the amount of pool tokens they want to sell, and the function should handle the exchange accordingly.

Impact: Users will swap the wrong amount of tokens,

Proof of Concept:

```
    function testIfSellPoolWorksCorrectly() public {
         vm.startPrank(liquidityProvider);
         console.log("weth.balanceOf(liquidityProvider)", weth.balanceOf
             (liquidityProvider));
         console.log("weth.balanceOf(address(pool))", weth.balanceOf(
             address(pool)));
         // poolToken.mint(liquidityProvider, 1 ether);
         weth.approve(address(pool), type(uint256).max);
         poolToken.approve(address(pool), type(uint256).max);
         pool.deposit(5 ether, 0, 10e18, uint64(block.timestamp));

         pool.sellPoolTokens(1e18);
         // Selling 1 pool token, this should be equal 0.5 ether that we
              should.
         vm.stopPrank();
         console.log("weth.balanceOf(liquidityProvider)", weth.balanceOf
             (liquidityProvider));
         console.log("weth.balanceOf(address(pool))", weth.balanceOf(
             address(pool)));
         console.log("poolToken.balanceOf(liquidityProvider)", poolToken
             .balanceOf(liquidityProvider));

         assertEq(weth.balanceOf(liquidityProvider), 0.5 ether);
     }
```

Recommended Mitigation:

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

Medium

[M-01] in `TSwapPool.sol::Deposit` `uint64 deadline` is never used causing transactions to be completed after the deadline!

Description: If a user deposits a transaction this can be executed at unexpected times as this doesn't have a deadline.

Impact: Transactions can be sent in unfavourable market conditions, `The deadline for the transaction to be completed by` This parameter is never used.

Proof of Concept: The `deadline` parameter is unused!

Recommended Mitigation:

```
1    function deposit(
2          uint256 wethToDeposit,
3          uint256 minimumLiquidityTokensToMint,
4          uint256 maximumPoolTokensToDeposit,
5          uint64 deadline
6      )
7          external
8  +       revertIfDeadlinePassed(deadline)
9          revertIfZero(wethToDeposit)
10         returns (uint256 liquidityTokensToMint)
11     }
```

[M-02] In the `PoolFactory:CreatePool` function, when generating a liquidity pool token with a symbol, it mistakenly utilizes `.name` instead of `.symbol`.

Description:

```
1          string memory liquidityTokenName = string.concat("T-Swap ",
               IERC20(tokenAddress).name());
2
3          string memory liquidityTokenSymbol = string.concat("ts", IERC20
               (tokenAddress).name());
```

Impact: Returns the .name instead of the .symbol it needed to return so it will have the wrong token-Symbol.

Recommended Mitigation:

```
1          string memory liquidityTokenName = string.concat("T-Swap ",
               IERC20(tokenAddress).name());
2
```

```
3 -      string memory liquidityTokenSymbol = string.concat("ts", IERC20(
      tokenAddress).name());
4
5 +      string memory liquidityTokenSymbol = string.concat("ts", IERC20(
      tokenAddress).symbol());
```

[M-03] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant.

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

Lows

[L-01] in `TSwapPool:SwapExactInput` uint256 `output` returned value is incorrect, it does not return anything.

Description: The `SwapExactInput` function is expected to return the actual amount of tokens bought by the caller, While it declares the output is not defined in the function.

```
1  function swapExactInput(
2          IERC20 inputToken,
3          uint256 inputAmount,
4          IERC20 outputToken,
5          uint256 minOutputAmount,
6          uint64 deadline
7      )
8          public
9          revertIfZero(inputAmount)
10         revertIfDeadlinePassed(deadline)
11         returns (
12
13             uint256 output
14         )
15     {
16         uint256 inputReserves = inputToken.balanceOf(address(this));
17         uint256 outputReserves = outputToken.balanceOf(address(this));
18
19         uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
               inputReserves, outputReserves);
20
21         if (outputAmount < minOutputAmount) {
```

```
22              revert TSwapPool__OutputTooLow(outputAmount,
                    minOutputAmount);
23          }
24
25          _swap(inputToken, inputAmount, outputToken, outputAmount);
26      }
```

Impact: Return value will always be zero, providing incorrect information to the frontend and user.

Proof of Concept:

In this proof of concept (edited) from the original swap, you will see the log returning the value 0. as the uint256 is never being defined in the function.

```
1   function testDepositSwap() public {
2           vm.startPrank(liquidityProvider);
3           weth.approve(address(pool), 100e18);
4           poolToken.approve(address(pool), 100e18);
5           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
6           vm.stopPrank();
7
8           vm.startPrank(user);
9           poolToken.approve(address(pool), 10e18);
10          // After we swap, there will be ~110 tokenA, and ~91 WETH
11          // 100 * 100 = 10,000
12          // 110 * ~91 = 10,000
13          uint256 expected = 9e18;
14
15          uint256 returnValue = pool.swapExactInput(poolToken, 10e18,
                weth, expected, uint64(block.timestamp));
16
17          console.log("ReturnValue", returnValue);
18      }
```

Recommended Mitigation:

```
1     function swapExactInput(
2           IERC20 inputToken,
3           uint256 inputAmount,
4           IERC20 outputToken,
5           uint256 minOutputAmount,
6           uint64 deadline
7       )
8           public
9           revertIfZero(inputAmount)
10          revertIfDeadlinePassed(deadline)
11  -        returns (uint256 output)
12  +        returns (uint256 outputAmount)
13      {
14          uint256 inputReserves = inputToken.balanceOf(address(this));
15          uint256 outputReserves = outputToken.balanceOf(address(this));
```

```
16
17          uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
                inputReserves, outputReserves);
18
19          if (outputAmount < minOutputAmount) {
20              revert TSwapPool__OutputTooLow(outputAmount,
                    minOutputAmount);
21          }
22
23          _swap(inputToken, inputAmount, outputToken, outputAmount);
24      }
```

[L-02] `Event:LiquidityAdded` is emittted backwards, which causes the wrong information to be emitted as an event.

Description:

Impact: In this scenario, the impact may not be significant, but it will display the liquidity added incorrectly, leading to confusion. Specifically, the variable “wethDeposited” is presented as “poolTokensToDeposit” and vice versa, which can cause misunderstanding.

Proof of Concept:

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

Recommended Mitigation:

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

[L-03] If an event has more than 3 events,it should be indexed, this will allow data to be more avaliable to offchain-tools.

Impact: This will not have a big impact, but it will make it more avaliable to tools when the events are indexed correctly.

Recommended Mitigation:

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

```
-    event LiquidityRemoved(address indexed liquidityProvider, uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
-    event Swap(address indexed swapper, IERC20 tokenIn, uint256 amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
+ event LiquidityAdded(address indexed liquidityProvider, uint256 indexed wethDeposited, uint256 indexed poolTokensDeposited);
+  event LiquidityRemoved(address indexed liquidityProvider, uint256 indexed wethWithdrawn, uint256 indexed poolTokensWithdrawn);
+    event Swap(address indexed swapper, IERC20 tokenIn, uint256 indexed amountTokenIn, IERC20 tokenOut, uint256  indexed amountTokenOut);
```

[L-04] In `TSwapPool.sol:SwapExactInput` has no natspec.

Description:

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

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

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

        if (outputAmount < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
        }

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

When not having any natspec, it makes it hard for other developers/auditors to understand the code

Recommended Mitigation: Add natspec/

Informationals

[I-01] In `TSwapPool.sol:Deposit` `uint256 poolTokenReserves` is never used in the function again.

Description:

```
1       if (totalLiquidityTokenSupply() > 0) {
2               uint256 wethReserves = i_wethToken.balanceOf(address(this))
                    ;
3
4               uint256 poolTokenReserves = i_poolToken.balanceOf(address(
                    this));
```

Impact: This will make calling the function more expensive.

Proof of Concept:

Recommended Mitigation:

Remove `uint256 poolTokenReserves` as this will reduce gas cost when calling the function.

[I-02] in `PoolFactory` error `error PoolFactory__PoolDoesNotExist(address tokenAddress);` is not being used.

Description:

Inside `PoolFactory` the `error PoolFactory__PoolDoesNotExist(address tokenAddress);` is never being used essentialy wasting gas,

Impact: Gas waste,

Proof of Concept:

-

Recommended Mitigatison: Use or remove to lower the gas.

```
1       function getPool(address tokenAddress) external view returns (
            address) {
2           return s_pools[tokenAddress];
3       }
```

Potentially can be used here when there are no pools being used, but it is not a must.