Audit Report TSwap Protocol

Version 1.0

AGMASO Security Reviews

October 19, 2024

Audit Report TSwap Protocol

AGMASO Security Reviews

19. October 2024

Prepared by: AGMASO Security Reviews Lead Auditors: Alejandro Guindo

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Scope Details
  - Actors / Roles
  - Issues found
- Findings
- High
  - TSwap SC
  - [H-1] Critical: Private function `TSwap::_swap` breaks the Core Invariant.
  - [H-2] Param `deadline` is not used in the `TSwap::deposit` allowing to execute the function even if the deadline has passed in time.
  - [H-3] Wrong Math in `TSwap::getInputAmountBasedOnOutput` breaks the Invariant of the Protocol and steal money from the User who is swapping and give than money to the fee taker.
  - [H-4] Missing Slippage protecction in `TSwap::swapExactOutput`, this could lead to wrong undesired swaps.
  - [H-5] `TSwap::sellPoolTokens` is using the wrong Function `swapExactOutput`, causing giving back to the Client an unexpected amount of Tokens the swap.

- Medium
  - PoolFactory SC
  - [M-1] Everyone can call `PoolFactory::createPool` and creates a new pool with WeirdERC20, creating a potential future issue in the protocol.
- Low
  - PoolFactory SC
  - [L-1] Missing zero address validation in the constructor.
  - TSwap SC
  - [L-1] Missing zero address validation in the constructor.
  - [L-2]: Follow the CEI to avoid Re-entrancy issues in the `TSwap::deposit` even if you are calling a internal function.
  - [L-3] Inside `TSwap::_addLiquidityMintAndTransfer` there is a Event with wrong order of arguments, causing issues in front-ends and confusion
  - [L-4] `TSwap::swapExactInput` is not returning anything but it should return `uint256 output`.
- Informational
  - PoolFactory SC
  - [I-1]: Event is missing `indexed` fields
  - [I-2]: PUSH0 is not supported by all chains
  - [I-3]: Unused Custom Error
  - [I-4] Mistake when setting the `liquidityTokenSymbol` string, causing a wrong Symbol of the LP Token
  - TSwap SC
  - [I-1]: **`public`** functions not used internally could be marked `external`
  - [I-2]: Define and use `constant` variables instead of using literals
  - [I-3]: Large literal values multiples of 10000 can be replaced with scientific notation
- Gas
  - TSwap SC
  - [G-1]: Dead code line. Elimitate this line to save GAS

Protocol Summary

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:

User A has 10 USDC They want to use it to buy DAI They swap their 10 USDC -> WETH in the USDC/WETH pool 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.

Disclaimer

The YOUR_NAME_HERE 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 Scope Details

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

- In Scope:

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

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

Actors / 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.

Issues found

Severity	Number of issues found
High	5
Medium	1
Low	5
Info	7
Gas	1
Total	19

Findings

High

TSwap SC

[H-1] Critical: Private function TSwap::_swap breaks the Core Invariant.

Description

Inside of the `TSwap::_swap` there is a piece of code that breaks the core invariant of the protocol x * y = k

```
1
2           swap_count++;
3   @>      if (swap_count >= SWAP_COUNT_MAX) {
4               swap_count = 0;
5               outputToken.safeTransfer(msg.sender, 1
                    _000_000_000_000_000_000);
6           }
```

There is a `swap_count` that is incrementing for each swap that happens. When the `swap_count` meets the `SWAP_COUNT_MAX` which has a value of 10, the protocol will reward to the client with `1_000_000_000_000_000_000` wei of Weth equal to 1 Weth. This reward is breaking then the core Invariant as it breaks the balance between `x` and `y`.

Impact

The impact is critical, as will break the core invariant and will cause a collapse of the Protocol. The likelihood is High, as will happen evey 10 swaps the 100% of the times. In addition to the break of Invariant, this architecture is vulnerable to MEV, as miners or other actors, can track this SC, and check when the exactly an user has sent to the blockchain the Transaction that if executed , will get the reward. A miner or bad actor , could see this when the Tx is in the mempool, and replicate the same Call adding a bigger TipFee to push his order before the one from the client, and finally steal the reward of the client.

Proof of Concepts

Poc

To prove this Concept, we have create a advanced suite of Stateful Fuzzing Test.

```
1
2   // SPDX-License-Identifier: MIT
3   pragma solidity 0.8.20;
```

```
4  import {Test, console2} from "forge-std/Test.sol";
5  import {StdInvariant} from "forge-std/StdInvariant.sol";
6  import {ERC20Mock} from "@openzeppelin/contracts/mocks/token/ERC20Mock.
       sol";
7  import {PoolFactory} from "../../src/PoolFactory.sol";
8  import {TSwapPool} from "../../src/TSwapPool.sol";
9  import {Handler} from "./Handler.t.sol";
10
11 contract InvariantTest is StdInvariant, Test {
12     ERC20Mock wethMock;
13     ERC20Mock poolToken;
14     PoolFactory poolFactory;
15     TSwapPool tSwapPool1;
16     Handler handler;
17     address addrTSwapPool1;
18     string liquidityTokenName = "LiquidityToken";
19     string liquidityTokenSymbol = "LP";
20
21     int256 constant STARTING_X = 100e18;
22     int256 constant STARTING_Y = 50e18;
23
24     function setUp() public {
25         //creates mocks
26         poolToken = new ERC20Mock(); //any ERC20
27         wethMock = new ERC20Mock(); //weth
28
29         //creates Factory and Factory creates new Pool
30         poolFactory = new PoolFactory(address(wethMock));
31         addrTSwapPool1 = poolFactory.createPool(address(poolToken));
32         tSwapPool1 = TSwapPool(addrTSwapPool1);
33
34         //Mint Tokens to this address(Test address)
35         poolToken.mint(address(this), uint256(STARTING_X));
36         wethMock.mint(address(this), uint256(STARTING_Y));
37
38         //Approve the TSwap SC
39         poolToken.approve(address(tSwapPool1), type(uint256).max);
40         wethMock.approve(address(tSwapPool1), type(uint256).max);
41
42         //Deposit to the Pool
43         // The pool is not warmup, means that the pool is empty without
               Ratio
44         // We create the Ratio thanks to a If() in the function that
              allow us to do so.
45         //We get 100% of the LP for the moment and will be equal to
              wethToDeposit Tokens
46         tSwapPool1.deposit(
47             uint256(STARTING_Y), //wethToDeposit
48             uint256(STARTING_Y), //minimum LP tokes to mint
49             uint256(STARTING_X), //maximum PoolTOkes to Deposit
50             uint64(block.timestamp) //deadline
```

```
        );

        handler = new Handler(tSwapPool1, poolFactory, wethMock,
            poolToken);

        bytes4[] memory selectors = new bytes4[](2);
        selectors[0] = handler.deposit.selector;
        selectors[1] = handler.swapPoolTokenforWethBasedOnOutputWeth.
            selector;
        targetSelector(
            FuzzSelector({addr: address(handler), selectors: selectors
                })
        );

        targetContract(address(handler));
    }

    function statefulFuzz_Holding() public {
        assertEq(handler.actualDeltaX(), handler.expectedDeltaX());
    }
    //Demonstrate the issue breaking the variant
    function statefulFuzz_HoldFormulaY() public {
        assertEq(handler.actualDeltaY(), handler.expectedDeltaY());
    }
}
```

Here is the Handler Sc, which wrapp the TSwap contract to bound the randomness of calling functions

```

// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;

import {Test, console} from "forge-std/Test.sol";
import {StdInvariant} from "forge-std/StdInvariant.sol";

import {ERC20Mock} from "@openzeppelin/contracts/mocks/token/ERC20Mock.
    sol";
import {PoolFactory} from "../../src/PoolFactory.sol";
import {TSwapPool} from "../../src/TSwapPool.sol";

contract Handler is Test {
    ERC20Mock wethMock;
    ERC20Mock poolToken;
    PoolFactory poolFactory;
    TSwapPool tSwapPool1;
    string liquidityTokenName = "LiquidityToken";
    string liquidityTokenSymbol = "LP";

    int256 public startingY;
    int256 public startingX;
```

```
22      int256 public expectedDeltaY;
23      int256 public expectedDeltaX;
24      int256 public actualDeltaY;
25      int256 public actualDeltaX;
26
27      address public LIQUIDITY_PROVIDER = makeAddr("LIQUIDITY_PROVIDER");
28      address public USER = makeAddr("USER");
29
30      constructor(
31          TSwapPool _tSwapPool1,
32          PoolFactory _poolFactory,
33          ERC20Mock _wethMock,
34          ERC20Mock _poolToken
35      ) {
36          tSwapPool1 = _tSwapPool1;
37          poolFactory = _poolFactory;
38          wethMock = ERC20Mock(_tSwapPool1.getWeth()); //We can pull it
                from Fn .getWeth();
39          poolToken = ERC20Mock(_tSwapPool1.getPoolToken()); //We can
                pull it from Fn .getPoolToken();
40
41          //We can also just send from the InvariantTest
42          //      wethMock = _wethMock ;
43          //     _poolToken = _poolToken;
44      }
45
46      function deposit(uint256 _wethToDeposit) public {
47          //Minimun that the user has to deposit
48          uint256 minWeth = tSwapPool1.getMinimumWethDepositAmount();
49          uint256 wethToDeposit = bound(
50              _wethToDeposit,
51              minWeth,
52              type(uint64).max
53          );
54
55          //Actual real starting point for both of the tokens of the pool
56          startingY = int256(wethMock.balanceOf(address(tSwapPool1)));
57          startingX = int256(poolToken.balanceOf(address(tSwapPool1)));
58
59          //We create State Variable "expectedDeltaY" to track all the
                way trhoug the Weth amount in the Handler
60          expectedDeltaY = int256(wethToDeposit);
61
62          //Calculating the "expectedDeltaX" thanks to the Fn
                getPoolTokensToDepositBasedOnWeth()
63          expectedDeltaX = int256(
64              tSwapPool1.getPoolTokensToDepositBasedOnWeth(
65                  uint256(expectedDeltaY)
66              )
67          );
68
```

```
69          //Starting parameters for deposit
70
71          //mint
72          vm.startPrank(LIQUIDITY_PROVIDER);
73          wethMock.mint(LIQUIDITY_PROVIDER, wethToDeposit);
74          poolToken.mint(LIQUIDITY_PROVIDER, uint256(expectedDeltaX));
75
76          //Approve the TSwap SC
77          wethMock.approve(address(tSwapPool1), type(uint256).max);
78          poolToken.approve(address(tSwapPool1), type(uint256).max - 1);
79
80          //deposit
81
82          tSwapPool1.deposit(
83              wethToDeposit,
84              0, //we can let this empty
85              uint256(expectedDeltaX),
86              uint64(block.timestamp)
87          );
88          vm.stopPrank();
89
90          //Actual real ending point for both of the tokens of the pool
                after deposit has happened
91          uint256 endingY = wethMock.balanceOf(address(tSwapPool1));
92          uint256 endingX = poolToken.balanceOf(address(tSwapPool1));
93
94          // Calculate the Delta. Also menas the change of the Y and X.
                Also has to be int256 becuase it could be negative
95          actualDeltaY = int256(endingY) - int256(startingY);
96          console.log("actualDeltaY", actualDeltaY);
97          actualDeltaX = int256(endingX) - int256(startingX);
98          console.log("actualDeltaX", actualDeltaX);
99      }
100
101     function swapPoolTokenforWethBasedOnOutputWeth(
102         uint256 outputWethAmount
103     ) public {
104         if (
105             wethMock.balanceOf(address(tSwapPool1)) <=
106             tSwapPool1.getMinimumWethDepositAmount()
107         ) {
108             return;
109         }
110         //Minimun that the user has to deposit
111         uint256 minWeth = tSwapPool1.getMinimumWethDepositAmount();
112         outputWethAmount = bound(outputWethAmount, minWeth, type(uint64
                ).max);
113
114         // We return and skip this test if the amount to swap is equal
                or more than the pool balances
115         if (outputWethAmount >= wethMock.balanceOf(address(tSwapPool1))
```

```
            ) {
116             return;
117         }
118
119         uint256 poolTokenAmount = tSwapPool1.
                getInputAmountBasedOnOutput(
120             outputWethAmount, // outputAmount
121             poolToken.balanceOf(address(tSwapPool1)), // inputReserves
122             wethMock.balanceOf(address(tSwapPool1)) // outputReserves
123         );
124         if (poolTokenAmount > type(uint64).max) {
125             return;
126         }
127
128         // We * -1 since we are removing WETH from the system
129         //Actual real starting point for both of the tokens of the pool
130         startingY = int256(wethMock.balanceOf(address(tSwapPool1)));
131         startingX = int256(poolToken.balanceOf(address(tSwapPool1)));
132
133         //We create State Variable "expectedDeltaY" to track all the
                way trhoug the Weth amount in the Handler
134         expectedDeltaY = int256(-1) * int256(outputWethAmount);
135
136         //Calculating the "expectedDeltaX" thanks to the Fn
                getPoolTokensToDepositBasedOnWeth()
137         expectedDeltaX = int256(poolTokenAmount);
138
139         // Mint any necessary amount of pool tokens
140         if (poolToken.balanceOf(USER) < poolTokenAmount) {
141             poolToken.mint(
142                 USER,
143                 poolTokenAmount - poolToken.balanceOf(USER) + 1
144             );
145         }
146
147         vm.startPrank(USER);
148         // Approve tokens so they can be pulled by the pool during the
                swap
149         poolToken.approve(address(tSwapPool1), type(uint256).max);
150
151         // Execute swap, giving pool tokens, receiving WETH
152         tSwapPool1.swapExactOutput({
153             inputToken: poolToken,
154             outputToken: wethMock,
155             outputAmount: outputWethAmount,
156             deadline: uint64(block.timestamp)
157         });
158
159         vm.stopPrank();
160
161         //updating ending variables
```

```
162         //Actual real ending point for both of the tokens of the pool
                after deposit has happened
163         uint256 endingY = wethMock.balanceOf(address(tSwapPool1));
164         uint256 endingX = poolToken.balanceOf(address(tSwapPool1));
165
166         // Calculate the Delta. Also menas the change of the Y and X.
                Also has to be int256 becuase it could be negative
167         actualDeltaY = int256(endingY) - int256(startingY);
168         actualDeltaX = int256(endingX) - int256(startingX);
169     }
170 }
```

Finally you can check the Logs:

Recommended mitigation

To mitigate this critical issue, you will need to eliminate this part of code provoking the issue, or rethink the protocol to add this reward, in some way that it will not break the core Invariant.

```
 1
 2
 3 function _swap(
 4         IERC20 inputToken,
 5         uint256 inputAmount,
 6         IERC20 outputToken,
 7         uint256 outputAmount
 8     ) private {
 9         if (
10             _isUnknown(inputToken) ||
11             _isUnknown(outputToken) ||
12             inputToken == outputToken
13         ) {
14             revert TSwapPool__InvalidToken();
15         }
16
17 -        swap_count++;
18 -        if (swap_count >= SWAP_COUNT_MAX) {
19 -            swap_count = 0;
20 -            outputToken.safeTransfer(msg.sender, 1
       _000_000_000_000_000_000);
21 -        }
22         emit Swap(
23             msg.sender,
24             inputToken,
25             inputAmount,
26             outputToken,
27             outputAmount
28         );
29
30         inputToken.safeTransferFrom(msg.sender, address(this),
               inputAmount);
```

```
31          outputToken.safeTransfer(msg.sender, outputAmount);
32      }
```

[H-2] Param `deadline` is not used in the `TSwap::deposit` allowing to execute the function even if the deadline has passed in time.

Description

The `TSwap::deposit` has a Param called `deadline` to bound the time that this function can be executed. This is used to prevent the `TSwap::deposit` to be executed if the condition of the deadline param is not met.

In this case, the `deadline` param is required but never used inside the function, so this functionality is not working and opens a big vulnerability with high likelihood, where the user can deposit even if he doesn't had the intention to do it.

```

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

Impact

This param is used to bound the time where the user wants to execute the function `TSwap::deposit`. The impact could be undesired deposit to the TSwap pool. The likelihood is hihg as will always happen for every call to this function.

Proof of Concepts

```

function test_auditDeadlineNotUsed() public {
        //!Deadline should create a bounding of time till the Fn could
           be executed. Here we proved that is not in use
        //! and allow to execute the FN always, doesn't matter the
           Deadline.
        vm.warp(1641070800);
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        uint256 actualTimeStamp = block.timestamp;
```

```
        uint256 deadline = block.timestamp - 5;
        console.log(actualTimeStamp);
        uint256 liquidityTokensToMint = pool.deposit(
            100e18,
            100e18,
            100e18,
            uint64(deadline)
        );
        console.log(liquidityTokensToMint);
        //!Function went through but becuase dead line is not working.
    }
```

Recommended mitigation

Add the same modifier that you are adding in other functions.

```

modifier revertIfDeadlinePassed(uint64 deadline) {
        if (deadline < uint64(block.timestamp)) {
            revert TSwapPool__DeadlineHasPassed(deadline);
        }
        _;
    }
```

This means:

```

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

[H-3] Wrong Math in `TSwap::getInputAmountBasedOnOutput` breaks the Invariant of the Protocol and steal money from the User who is swapping and give than money to the fee taker.

Description

Inside `TSwap::getInputAmountBasedOnOutput` a wrong division is creating a huge fee for the Fee Taker when in theory should be as big as 0.3% of the Swap operation.

```

function getInputAmountBasedOnOutput(
```

```
3           uint256 outputAmount,
4           uint256 inputReserves,
5           uint256 outputReserves
6       )
7           public
8           pure
9           revertIfZero(outputAmount)
10          revertIfZero(outputReserves)
11          returns (uint256 inputAmount)
12      {
13          return
14  @>              ((inputReserves * outputAmount) * 10000) /
15              ((outputReserves - outputAmount) * 997);
16
17      }
```

The number should be `1000` but here in the protocol we have `10000`.

Impact

This has a huge impact as it will break the Invariant `x * y = k` and in addition will steal money from the client using the swap method. The likelihood is 100% of the times we call this function. That is why we give a High severity vulnerability.

Recommended mitigation

You can avoid this issues by using Magil Numbers. If you create `constant` of this number where you called for instance `PRECISION` , then you would have avoided this typo error that creates a High vulnerability.

You can just correct the number.

```
1
2   + uint256 public constant PRECISION = 1000;
3   .
4   .
5   .
6   .
7   function getInputAmountBasedOnOutput(
8           uint256 outputAmount,
9           uint256 inputReserves,
10          uint256 outputReserves
11      )
12          public
13          pure
14          revertIfZero(outputAmount)
15          revertIfZero(outputReserves)
16          returns (uint256 inputAmount)
17      {
18          return
19  -           ((inputReserves * outputAmount) * 10000) /
```

```
20 +             ((inputReserves * outputAmount) * PRECISION)
21               ((outputReserves - outputAmount) * 997);
22
23     }
```

[H-4] Missing Slippage protecction in `TSwap::swapExactOutput`, this could lead to wrong undesired swaps.

Description

Inside of the `TSwap::swapExactOutput` function we are missing slippage check.

```
1
2  function swapExactOutput(
3          IERC20 inputToken,
4          IERC20 outputToken,
5          uint256 outputAmount,
6          uint64 deadline
7      )
8          public
9          revertIfZero(outputAmount)
10         revertIfDeadlinePassed(deadline)
11         returns (uint256 inputAmount)
12     {
13         uint256 inputReserves = inputToken.balanceOf(address(this));
14         uint256 outputReserves = outputToken.balanceOf(address(this));
15
16         inputAmount = getInputAmountBasedOnOutput(
17             outputAmount,
18             inputReserves,
19             outputReserves
20         );
21 @>
22         _swap(inputToken, inputAmount, outputToken, outputAmount);
23     }
```

We are missing a piece of code such the one you use in `TSwap::swapExactInput`:

```
1
2   if (outputAmount < minOutputAmount) {
3              revert TSwapPool__OutputTooLow(outputAmount,
                  minOutputAmount);
4          }
```

This will assure that we are not executing non desired swaps

Impact

The lack of slippage protection in the TSwap::swapExactOutput function means that the swap may

execute at a price significantly different from the expected one. This could result in users receiving fewer tokens than anticipated or paying more than they intended. The absence of slippage checks can expose the protocol to market manipulation, create inefficiencies in swaps, and lead to potential user dissatisfaction, especially in volatile markets.

The impact includes:

User losses: Users may end up with a less favorable trade than expected, leading to losses. Market manipulation risk: In extreme cases, attackers could manipulate reserves and perform malicious swaps, affecting liquidity. Reputation risk: If users experience frequent slippage issues, it could damage the platform's reputation and result in loss of trust. Economic inefficiency: Failing to account for slippage could lead to suboptimal trading conditions, reducing the effectiveness of liquidity pools.

Proof of Concepts

Recommended mitigation

To mitigate the risk of undesired swaps due to slippage, you should implement a check to ensure that the outputAmount meets or exceeds a minimum acceptable amount (minOutputAmount). If the actual output is less than this minimum, the transaction should revert to prevent the swap from executing under unfavorable conditions. This safeguard is crucial, especially in markets with high volatility.

You can implement this as follows:

Add slippage check: In the TSwap::swapExactOutput function, add a check to ensure that the output amount is greater than or equal to the minOutputAmount provided by the user.

```
1
2  function swapExactOutput(
3          IERC20 inputToken,
4          IERC20 outputToken,
5          uint256 outputAmount,
6          uint64 deadline,
7  +        uint256 minOutputAmount
8      )
9          public
10         revertIfZero(outputAmount)
11         revertIfDeadlinePassed(deadline)
12         returns (uint256 inputAmount)
13     {
14         uint256 inputReserves = inputToken.balanceOf(address(this));
15         uint256 outputReserves = outputToken.balanceOf(address(this));
16
17         inputAmount = getInputAmountBasedOnOutput(
18             outputAmount,
19             inputReserves,
20             outputReserves
21         );
22
```

```
23 +         // Slippage protection: Ensure output is not below acceptable
       level
24 +         if (outputAmount < minOutputAmount) {
25 +             revert TSwapPool__OutputTooLow(outputAmount,
       minOutputAmount);
26 +         }
27
28           _swap(inputToken, inputAmount, outputToken, outputAmount);
29       }
```

[H-5] TSwap::sellPoolTokens is using the wrong Function swapExactOutput, causing giving back to the Client an unexpected amount of Tokens the swap.

Description

The function `TSwap::sellPoolTokens` is supposed to be used to sell pooltokens and get in return Weth. It is just a wrapper of the Swap. In this case `TSwap::sellPoolTokens` just need one parameter, the `uint256 poolTokenAmount` which are the amount of PoolTokens that we want to exchange for Weth. Logically, if we only have this parameter avalaible, we need to use the function `swapExactInput` but in this case the protocol is using `swapExactOutput` which make no sense and will break the protocol.

Impact

It has big impact for the User, as he will get an unexpected amount of Weth Tokens in return. In addition, will make taht the protocol is not working as expected even if it will not break the Invariant. The likelihood is High as every call to the function will act the same and anyone could called.

Recommended mitigation

To mitigate this issue , you just need to change the `swapExactOutput` to the `swapExactInput`.

```
1
2  function sellPoolTokens(
3          uint256 poolTokenAmount
4      ) external returns (uint256 wethAmount) {
5          return
6          // @audit-High: it should use swapExactInput, as we are giving
               only as paramter the tokens pools taht we want to sell
7
8  -           swapExactOutput(
9  -               i_poolToken,
10 -               i_wethToken,
11 -               poolTokenAmount,
12 -               uint64(block.timestamp)
13 -           );
14 +           swapExactInput(
```

```
15 +                 i_poolToken,
16 +                 poolTokenAmount,
17 +                 i_wethToken,
18 +                 minOutputAmount,
19 +                 deadline
20 +             )
21
22      }
```

Medium

PoolFactory SC

[M-1] Everyone can call `PoolFactory::createPool` and creates a new pool with WeidERC20, creating a potential future issue in the protocol.

Description

In the `PoolFactory::createPool` there isn't a restriction of tokenaddress that you could add to create a new pool. In addition the function is external without resticcions, so anyone can create a pool.

This lead to a potential future vulnerability, as a user could create a pool with a Weird ERC20 that could work differently as the protocol had planned and break the protocol in some point.

For instance check line 59:

```
1 string memory liquidityTokenName = string.concat(
2             "T-Swap ",
3             IERC20(tokenAddress).name()
4         );
```

If the weird ERC20 revert on calling `IERC20(tokenAddress).name()` will revert also the creation of the entire pool, leading to a bad function of the protocol.

Impact

The impact is Medium , If you allow an incompatible token (such as a "WeirdERC20") to create a pool, and that token has unexpected or faulty behavior, it can break the contract's logic and cause pool creation to fail. This could result in a loss of functionality on the platform, preventing the creation of pools with valid tokens, or lead to users interacting with defective pools, which could negatively impact the protocol's reputation and usability.

Proof of Concepts

PoC

We create a WeirdERC20 where it reverts when function `name()` is called. This reverts all the creation of the pool

```
contract ERC20Weird is Context, IERC20, IERC20Metadata, IERC20Errors {
    mapping(address account => uint256) private _balances;

    mapping(address account => mapping(address spender => uint256))
        private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * All two of these values are immutable: they can only be set once
         during
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual returns (string memory) {
@>          revert();
    }
```

```
function test_auditWeird20Pool() public {
        vm.expectRevert();
        address poolAddress = factory.createPool(address(weird));
    }
```

Logs:

Ran 1 test for test/unit/PoolFactoryTest.t.sol:PoolFactoryTest [PASS] test_auditWeird20Pool() (gas: 15530) Traces: [15530] PoolFactoryTest::test_auditWeird20Pool() [0] VM::expectRevert(custom error f4844814:) ← [Return] [5362] PoolFactory::createPool(ERC20Weird: [0xc7183455a4C133Ae270771860664b6B7ec320bB1] [130] ERC20Weird::name() [staticcall] ← [Revert] EvmError: Revert ← [Revert] EvmError: Revert ← [Stop]

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.81ms (859.08µs CPU time)

Recommended mitigation

Whitelist of tokens: You could implement a whitelist mechanism for allowed tokens. Only tokens that are approved by the contract owner (or another trusted entity) should be able to create pools. This whitelist will ensure that only verified tokens can interact with the contract.

Token compatibility check: Before allowing the creation of a pool, you could add a validation function that checks whether the token meets the expectations of the ERC20 standard. This validation could verify whether the name(), symbol(), decimals(), etc. functions exist and do not fail when called.

Owner-based restriction (role-based): As you mentioned, a good practice would be to restrict pool creation to the contract owner or an account with a specific role, such as OWNER or a POOL_CREATOR role. This ensures that only trusted accounts can decide which tokens can be paired.

Low

PoolFactory SC

[L-1] Missing zero address validation in the constructor.

Description

Detect missing zero address validation.

```
1
2  constructor(address wethToken) {
3          // @audit-info Lack of zero-check
4          i_wethToken = wethToken;
5      }
```

Recommended mitigation

Check that the address is not zero.

```
1
2  constructor(address wethToken) {
3  +        if (wethToken == address(0)) {
4  +            revert PoolFactory__ZeroAddressNotAllowed(); // Lanza un
       error si la dirección es cero
5  +        }
6          i_wethToken = wethToken;
7      }
```

TSwap SC

[L-1] Missing zero address validation in the constructor.

Description

Detect missing zero address validation.

```
1
2  constructor(
3          address poolToken,
4          address wethToken,
5          string memory liquidityTokenName,
6          string memory liquidityTokenSymbol
7      ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
8
9  @>          // @audit-info Zero Address Check missing.
10         i_wethToken = IERC20(wethToken);
11         i_poolToken = IERC20(poolToken);
12     }
```

Recommended mitigation

Check that the address is not zero.

```
1
2  constructor(
3          address poolToken,
4          address wethToken,
5          string memory liquidityTokenName,
6          string memory liquidityTokenSymbol
7      ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
8
9  +        if (poolToken == address(0)) {
10 +            revert TSwap__ZeroAddressNotAllowed();
11 +        }
12 +        if (wethToken == address(0)) {
13 +            revert TSwap__ZeroAddressNotAllowed();
14 +        }
15         i_wethToken = IERC20(wethToken);
16         i_poolToken = IERC20(poolToken);
17     }
```

[L-2]: Follow the CEI to avoid Re-entrancy issues in the `TSwap::deposit` even if you are calling a internal function.

Description and Impact

Inside the `TSwap::deposit`, the protocol calls `_addLiquidityMintAndTransfer()` before the update of `liquidityTokensToMint = wethToDeposit;`. In this case, the `liquidityTokensToMint` is not a State Variable, meaning the impact will be Low and also `_addLiquidityMintAndTransfer()` is a internal function, so the risk is lower. But anyway, `_addLiquidityMintAndTransfer()` is performing external calls, so we need to follow the CEI to avoid Re-entrancy issues.

Recommended mitigation

To mitigate this issue do the following:

```
1
2  else {
3              // This will be the "initial" funding of the protocol. We
                  are starting from blank here!
4              // We just have them send the tokens in, and we mint
                  liquidity tokens based on the weth
5
6  +           liquidityTokensToMint = wethToDeposit;
7
8              _addLiquidityMintAndTransfer(
9                  wethToDeposit,
10                 maximumPoolTokensToDeposit,
11                 wethToDeposit
12             );
13 -            liquidityTokensToMint = wethToDeposit;
14         }
```

[L-3] Inside `TSwap::_addLiquidityMintAndTransfer` there is a Event with wrong order of arguments, causing issues in front-ends and confusion

Description

Inside `TSwap::_addLiquidityMintAndTransfer` there is a Event with wrong order of arguments.

```
1
2  function _addLiquidityMintAndTransfer(
3          uint256 wethToDeposit,
4          uint256 poolTokensToDeposit,
5          uint256 liquidityTokensToMint
6      ) private {
7          _mint(msg.sender, liquidityTokensToMint);
8          // @audit-Low: Wrong order of arguments
9  @>          emit LiquidityAdded(msg.sender, poolTokensToDeposit,
       wethToDeposit);
```

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

Impact

Probably it will mess with all the systems that use Events as a vital information to fire another actions. For instance Front-ends , etc…

Recommended mitigation

Correct the order of arguments

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

[L-4] `TSwap::swapExactInput` is not returning anything but it should return `uint256 output`.

Description

This output will always return ZERO, which is bad. Because if used somewhere else, it will create a High severity. Luckily is not used elsewher. For this reason we give low Impact, but high likelihood.

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
@>            uint256 output // @audit-Low. You are missing the return. OutputAmount should be the retunr. Change it.
        )
    // @audit-low:  This output will always return ZERO, which is bad. Because is not used elsewhere , we give low Impact, but high likelihood
    {
```

Recommended mitigation

To fix this issue you just need to do as following:

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

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

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

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

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

Informational

PoolFactory SC

[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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
    event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
    event Swap(
```

[I-2]: PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

2 Found Instances

- Found in src/PoolFactory.sol Line: 15

```
pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
pragma solidity 0.8.20;
```

[I-3]: Unused Custom Error

it is recommended that the definition be removed when custom error is unused

1 Found Instances

- Found in src/PoolFactory.sol Line: 22

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

[I-4] Mistake when setting the `liquidityTokenSymbol` string, causing a wrong Symbol of the LP Token

Description

Mistake when setting the `liquidityTokenSymbol` string, causing a wrong Symbol of the LP Token. You are contateniting `IERC20(tokenAddress).name()` which is wrong. It should be the symbol of the `IERC20(tokenAddress)` and not the name. It could lead to confusion.

```
1
2  // @audit-info: This should be .symbol()
3          string memory liquidityTokenSymbol = string.concat(
4              "ts",
5              IERC20(tokenAddress).name()
6          );
```

Recommended mitigation

```
1
2   string memory liquidityTokenSymbol = string.concat(
3              "ts",
4  -            IERC20(tokenAddress).name()
5  +            IERC20(tokenAddress).symbol()
6          );
```

TSwap SC

[I-1]: `public` functions not used internally could be marked `external`

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

1 Found Instances

- Found in src/TSwapPool.sol Line: 297

```
1      function swapExactInput(
```

[I-2]: Define and use `constant` variables instead of using literals

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

4 Found Instances

- Found in src/TSwapPool.sol Line: 274

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

- Found in src/TSwapPool.sol Line: 293

```
1              ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 445

```
1                  1e18,
```

- Found in src/TSwapPool.sol Line: 454

```
1                  1e18,
```

[I-3]: Large literal values multiples of 10000 can be replaced with scientific notation

Use `e` notation, for example: `1e18`, instead of its full numeric value.

3 Found Instances

- Found in src/TSwapPool.sol Line: 45

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

- Found in src/TSwapPool.sol Line: 292

```
1              ((inputReserves * outputAmount) * 10000) /
```

- Found in src/TSwapPool.sol Line: 393

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

Gas

TSwap SC

[G-1]: Dead code line. Elimitate this line to save GAS

1 Found Instances

- Found in src/TSwapPool.sol Line: 135

```
1          uint256 poolTokenReserves = i_poolToken.balanceOf(address(
               this));
```