Protocol Audit Report

Parth Sharma

March 23, 2023

Protocol Audit Report

Version 1.0

Parth

March 24, 2024

Protocol Audit Report

Parth Sharma

March 23, 2023

Prepared by: Tonchi Lead Auditors:

- Tonchi

Table of Contents

Disclaimer

Tonchi 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

Scope

-> src - TSwapPool.sol - PoolFactory.sol

Roles

- Liquidity Provider : Adds or remove liquidity to the pool
- Owner : Deploy the Swap protocol
- User : Make swap between weth and any ERC20 token.

Issues found

Severtity	Number of issues found
High	4
Medium	1
Low	2
Gas	2
Info	2
Total	11

Findings

HIGH

[H-1] Invariant (x multiple y = k) breaks, it sends massive amount to user

Description: In `TSwapPool::_swap` function, on every 10th transaction this function send a massive `1_000_000_000_000_000_000` amount to user. It is the severe disruption of invariants of the protocol which says `x*y=k`. 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.

Proof of Code `Add the following test suit: Add Invariants.t.sol:`

```solidity
//SPDX-License-Identifier: MIT

pragma solidity 0.8.20;

import { StdInvariant } from "lib/forge-std/src/StdInvariant.sol";
import { Test } from "lib/forge-std/src/Test.sol";
import { TSwapPool } from "../../../src/TSwapPool.sol";
import { PoolFactory } from "../../../src/PoolFactory.sol";
import { ERC20Mock } from "../mocks/ERC20Mock.sol";
import { Handler } from "./Handler.t.sol";

contract Invariant is StdInvariant, Test {
    PoolFactory public poolFactory;
    TSwapPool public pool;
    ERC20Mock public poolToken;
    ERC20Mock public weth;
    Handler handler;

    uint256 public constant STARTING_X_AMOUNT = 50e18; // Weth Token amount
    uint256 public constant STARTING_Y_AMOUNT = 100e18; // poolToken amount

    function setUp() external {
        weth = new ERC20Mock();
        poolToken = new ERC20Mock();

        poolFactory = new PoolFactory(address(weth));
        pool = TSwapPool(poolFactory.createPool(address(poolToken))); //createPool function
            // weth, and return TswapPool pool address.

        weth.mint(address(this), STARTING_X_AMOUNT);
        poolToken.mint(address(this), STARTING_Y_AMOUNT);
        weth.approve(address(pool), type(uint256).max);
        poolToken.approve(address(pool), type(uint256).max);
        // 115792089237316195423570985008687907853269984665640564039457584007913129639935
        pool.deposit(STARTING_X_AMOUNT, STARTING_X_AMOUNT, STARTING_Y_AMOUNT, uint64(block.

        handler = new Handler(pool);

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

    // This breaks our invariant
    function invariant_TestInvariantOfWethFollowsMaths() public {
        assertEq(handler.actualDeltaWethAmount(), handler.expectedDeltaWethAmount());
    }

    function invariant_TestInvariantOfPoolTokenFollowsMaths() public {
        assertEq(handler.actualDeltaPoolTokenAmount(), handler.expectedDeltaPoolTokenAmount
    }
}
```

Add Handler.t.sol:

```
//SPDX-License-Identifier: MIT

pragma solidity 0.8.20;

import { Test, console } from "lib/forge-std/src/Test.sol";
import { ERC20Mock } from "../mocks/ERC20Mock.sol";
import { TSwapPool } from "../../src/TSwapPool.sol";

contract Handler is Test {
    ERC20Mock public weth;
    ERC20Mock public poolToken;
    TSwapPool pool;

    // Ghost variables
    int256 public startingWethAmount;
    int256 public startingPoolTokenAmount;

    int256 public actualDeltaWethAmount;
    int256 public actualDeltaPoolTokenAmount;

    int256 public expectedDeltaWethAmount;
    int256 public expectedDeltaPoolTokenAmount;

    address public liquidityProvider = makeAddr("liquidityProvider");
    address public immutable user = makeAddr("user");

    constructor(TSwapPool _pool) {
        pool = _pool;
        weth = ERC20Mock(pool.getWeth());
        poolToken = ERC20Mock(pool.getPoolToken());
    }

    /**
     * @dev This function allows user to make swaps, and updates startingWethAmount, starti
     *  actualDeltaWethAmount, actualDeltaPoolTokenAmount, expectedDeltaWethAmount, expecte
     */
```

```
function swapPoolTokenForWethBasedOnOutputWeth(uint256 _outputWethAmount) public {
    if (weth.balanceOf(address(pool)) <= pool.getMinimumWethDepositAmount()) {
        return;
    }

    uint256 outputWethAmount =
        bound(_outputWethAmount, pool.getMinimumWethDepositAmount(), weth.balanceOf(addr
    if (outputWethAmount >= weth.balanceOf(address(pool))) {
        return;
    }

    uint256 inputPoolTokenAmount = pool.getInputAmountBasedOnOutput(
        outputWethAmount, poolToken.balanceOf(address(pool)), weth.balanceOf(address(poo
    );

    if (inputPoolTokenAmount > poolToken.balanceOf(user)) {
        poolToken.mint(user, inputPoolTokenAmount - poolToken.balanceOf(user) + 1);
    }

    startingWethAmount = int256(weth.balanceOf(address(pool)));
    startingPoolTokenAmount = int256(poolToken.balanceOf(address(pool)));

    expectedDeltaWethAmount = int256(outputWethAmount) * -1;
    expectedDeltaPoolTokenAmount = int256(inputPoolTokenAmount);

    vm.startPrank(user);
    poolToken.approve(address(pool), type(uint256).max);
    console.log("Weth balance of Weth token: ", weth.balanceOf(address(pool)));

    pool.swapExactOutput(poolToken, weth, outputWethAmount, uint64(block.timestamp));
    vm.stopPrank();

    int256 endingWethAmount = int256(weth.balanceOf(address(pool)));
    int256 endingPoolTokenAmount = int256(poolToken.balanceOf(address(pool)));

    actualDeltaWethAmount = endingWethAmount - startingWethAmount;
    actualDeltaPoolTokenAmount = endingPoolTokenAmount - startingPoolTokenAmount;
}

function deposit(uint256 wethAmountToDeposit) public {
    wethAmountToDeposit = bound(wethAmountToDeposit, pool.getMinimumWethDepositAmount()
    uint256 poolTokensToDeposit = pool.getPoolTokensToDepositBasedOnWeth(wethAmountToDep

    startingWethAmount = int256(weth.balanceOf(address(pool)));
    startingPoolTokenAmount = int256(poolToken.balanceOf(address(pool)));
```

```
        expectedDeltaWethAmount = int256(wethAmountToDeposit);
        expectedDeltaPoolTokenAmount = int256(poolTokensToDeposit);

        vm.startPrank(liquidityProvider);
        weth.mint(liquidityProvider, wethAmountToDeposit);
        poolToken.mint(liquidityProvider, poolTokensToDeposit);

        weth.approve(address(pool), wethAmountToDeposit);
        poolToken.approve(address(pool), poolTokensToDeposit);

        pool.deposit(wethAmountToDeposit, 0, poolTokensToDeposit, uint64(block.timestamp));
        vm.stopPrank();

        uint256 endingWethAmount = weth.balanceOf(address(pool));
        uint256 endingPoolTokenAmount = poolToken.balanceOf(address(pool));

        actualDeltaWethAmount = int256(endingWethAmount) - startingWethAmount;
        actualDeltaPoolTokenAmount = int256(endingPoolTokenAmount) - startingPoolTokenAmount
    }
}
```

In above test-suit `actualDeltaWethAmount` and `actualDeltaPoolTokenAmount` is different from `expectedDeltaWethAmount` and `expectedDeltaPoolTokenAmount`, so it doesn't follow `x*y=k` and breaks invariant of TSwap protocol.

Impact: Breaks invariant, which is against the intention of protocol. And sends huge amount to the user.

Proof of Concept:

1. The user maliciously swap 10 transaction to get incentives, which is a HUGE amount.
2. This way a user can drained all the money from the pool

Proof Of Code

Add this code block to your unit test suit:

```
  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;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);
        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();

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

Recommended Mitigation: Remove the extra big amount of money transaction in `TSwapPool::_swap`.

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

[H-2] `TSwapPool::getInputAmountBasedOnOutput` 10000 is used instead of 1000 it took way more fees from user

Description: `TSwapPool::getInputAmountBasedOnOutput` function have mathematical calculation where the protocol charges some `0.03%` fees from user to swap on TSwap protocol, it uses 10000 instead of 1000. As a consequence, the protocol charges way more fees (nearly about `91.5%`) from user than expected.

Proof of Code

```
    function testSwapTakesAppropriateFees() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 200e18);
        poolToken.approve(address(pool), 200e18);

        uint256 wethToDeposit = 50e18;
        uint256 lptokenToMint = 50e18;
        uint256 poolTokenToDeposit = 100e18;

        pool.deposit(wethToDeposit, lptokenToMint, poolTokenToDeposit, uint64(block.timestam
        vm.stopPrank();

        uint256 expectedPoolTokenToDeposit = 11e18;
        uint256 actualwethDeposit = 5e18;

        // user want to get 5 weth, How much pooltoken need to be deposited
        vm.startPrank(user);
        poolToken.mint(address(user), 120e18);
        poolToken.approve(address(pool), 120e18);
        uint256 startingUserBalancePoolToken = poolToken.balanceOf(user);
        uint256 startingUserBalanceWeth = weth.balanceOf(user);

        pool.swapExactOutput(poolToken, weth, actualwethDeposit, uint64(block.timestamp));
        vm.stopPrank();

        uint256 endingweth = weth.balanceOf(user);
        uint256 endingPoolTokenBalance = poolToken.balanceOf(user);
        uint256 actualPoolTokenDeposited = startingUserBalancePoolToken - endingPoolTokenBal

        assertEq(expectedPoolTokenToDeposit, actualPoolTokenDeposited);
        assertEq(startingUserBalanceWeth + 5e18, endingweth);
    }
```

It will give the following error:

```
[FAIL. Reason: assertion failed] testSwapTakesAppropriateFees() (gas: 286762)
Logs:
  Error: a == b not satisfied [uint]
        Left: 11000000000000000000
       Right: 111445447453471525688
```

Impact: Charges more fees than expected, huge loss of funds of user.

Proof of Concept: Call the function below:

1. Call `TSwapPool::swapExactOutput` as a user.
2. It calls `TSwapPool::getInputAmountBasedOnOutput`.
3. Than the amount deposited from user to pool is 10 times more than expected by protocol.

Recommended Mitigation: Do following correction in `TSwapPool::getInputAmountBasedOnOutput`:

```
-   return ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997
+   return ((inputReserves * outputAmount) * 1000) / ((outputReserves - outputAmount) * 997
```

[H-3] `TSwapPool::sellPoolTokens` send `poolTokenAmount` as parameter and call `TSwapPool::swapExactOutput` which recieve wethAmount as parameter, huge loss of funds of user

Description: `TSwapPool::sellPoolTokens` function is intended to take argument `poolTokenAmount` but our function calls `TSwapPool::swapExactOutput` and it takes `wethAmount` so here we mistakenly sell some poolToken amount from user in excange of weth amount which we pass, if we pass `poolTokenAmount = 112` than pool sends `wethAmount = 112` to user and pull poolToken amount according to excahnge rate.

@> NOTE : To run following test, In `TSwapPool::getInputAmountBasedOnOutput` correct 10000 to 1000 to make sure our calculation correct.

Proof of Code

Following is the test for `TSwapPool::sellPoolTokens`:

```
//POC for sellPoolTokens
    function testSellPoolTokenFunction() public {
        vm.startPrank(liquidityProvider);
        //maths
        // here we take 500 poolToken and 200 weth Token in pool.
        // whereas our actual liquidity amount should between 100 poolToken and 50 weth toke
        // we send 112 poolTokens, and think that we can get nearly some 36 weth token amoun
        // but we actually getting 112 wethAMount, and end up sending pooltokenamount accor
        // write to correct 10000 to 1000 to make sure our calculation correct.
        // poolTokenAmount = ((inputReserves * outputAmount) * 1000) / ((outputReserves - o
        // ((500*112)*1000)/ ((200 - 112)*997)
        // Here we losing 638 poolTOken amount instead of 112 poolToken amount that we think
        uint256 wethAmounttoDeposit = 300e18;
        uint256 poolTokenAmountToDeposit = 800e18;

        weth.mint(liquidityProvider, wethAmounttoDeposit);
        poolToken.mint(liquidityProvider, poolTokenAmountToDeposit);

        weth.approve(address(pool), wethAmounttoDeposit + 200e18);
        poolToken.approve(address(pool), poolTokenAmountToDeposit + 200e18);

        pool.deposit(wethAmounttoDeposit, wethAmounttoDeposit, poolTokenAmountToDeposit, uin
        vm.stopPrank();

        vm.startPrank(user);
        poolToken.mint(user, 1000e18);
        poolToken.approve(address(pool), 1000e18);
        // weth.approve(address(pool), 120e18);

        // User weth amount balance is 130, and poolToken amount is 10.
        // according to natspec we should sell poolToken in poolTOkenAmount. but actually o
        // poolTokenAmount

        uint256 startingPoolTokenAmountUser = poolToken.balanceOf(address(user));
        uint256 expectedPoolTokenAmountUser = startingPoolTokenAmountUser - 112e18;

        uint256 poolTokenAmountActuallyWethAmount = 112e18;
        pool.sellPoolTokens(poolTokenAmountActuallyWethAmount);

        uint256 endingPoolTokenAMountUser = poolToken.balanceOf(address(user));

        uint256 actualPoolTokenAmountUser = startingPoolTokenAmountUser - endingPoolTokenAM
        vm.stopPrank();

        assertEq(actualPoolTokenAmountUser, expectedPoolTokenAmountUser);
        // poolTokenAmountToSell = 112.000000000000000000
        // actualPoolTokenAmountSold = 1000 - actualPoolTokenAmount = 532.0000000000000000
        // Huge loss of funds.
        // actualPoolTokenAmount = 478.029834183401267632
        // expectedPoolTokenAmountUser = 898.000000000000000000
    }
```

Impact: User loss huge amount of poolToken funds than expected to sell.

Proof of Concept: Following comments are explainer:

- `suppose we take 500 poolToken and 200 weth Token in pool.`
- `whereas our actual liquidity provided amount should be 1010 poolToken and 500 weth t`
- `we send 112 poolTokens, and think that we can get nearly some 36 weth token amount`
- `but we actually getting 112 wethAMount, and end up sending pooltokenamount according`
- `write to correct 10000 to 1000 to make sure our calculation correct.`
- `function ```TSwapPool::sellPoolTokens``` call ```swapExactOutput``` which calls ```g`
- `poolTokenAmount = ((inputReserves * outputAmount) * 1000) / ((outputReserves - outpu`
- `((500*112)*1000)/ ((200 - 112)*997)`
- `Here we losing 638 poolTOken amount instead of 112 poolToken amount that we think.`

Recommended Mitigation: Following are the ways to mitigate:

1. Adds and remove the following lines. In this we send that OutputwethAmount we want take the InputPoolTokenAmount to input.

```
-function sellPoolTokens(uint256 poolTokenAmount) external returns (uint256 wethAmount) {
-         return swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount, uint64(block.time
```

```
+function sellPoolTokens(uint256 OutputwethAmount) external returns (uint256 InputPoolToken
+        return swapExactOutput(i_poolToken, i_wethToken, OutputwethAmount, uint64(block.ti
```

2. We should call `TSwapPool::swapExactInput` inside `TSwapPool::sellPoolToken` function, here we send input poolToken amount and get weth amount simply don't need to change parameters here.

[H-4] Lack of slippage protection in `TSwapPool::swapExactOutput`, causes user to receive way fewer tokens

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

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

Proof of Concept:

1. The price of 1 weth is now 1000 poolTokens
2. User invoked `swapExactOutput` for 1 weth.
    1. inputToken = poolToken
    2. outputToken = weth
    3. outputAmount = 1
    4. deadline = whatever
3. The function doesn't offer a `maxInputAmount`
4. As the transaction is in pending condition in mempool, and the market changes! the price moves huge 1 weth ->10000 poolToken. 10x more than user expected.
5. The transaction complete and user sent the protocol 10,000 poolToken instead of 1000.

Recommended Mitigation: Add the following code block in `TSwapPool::swapExactOutput` :

```
function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
+        uint256 maxInputAmount,
        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, outputReserve
+        if(inputAmount > maxInputAmount){
+            revert();
+        }
```

MEDIUM

[M-1] `TSwapPool::deposit` deadline is not used, causing transactions to complete even after the deadline

Description: `TSwapPool::deposit` accepts deadline parameter, which is intended to "The deadline of the parameter to be completed by". This parameter is never used. As a consequence, operations adds liquidity to the pool might be executed at unexpected times, where market conditions for deposit are not favourable.

Impact: Transaction could be sent when market condition are unfavourable to deposit.

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

Recommended Mitigation: Adds the following recommendation:

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

LOW

[L-1] In `TSwapPool::_addLiquidityMintAndTransfer` function, parameter passed in wrong order

Description: In `TSwapPool::_addLiquidityMintAndTransfer` function, parameter passed in `LiquidityAdded` event are not incorrect order which can mislead the user sees transaction data off-chain. And It sends wrong information off-chain.

Impact: sends wrong information to off-chain user about transaction, which further confuse one which amount to send or recieve.

Recommended Mitigation: Do following changes in `TSwapPool::_addLiquidityMintAndTransfer` function:

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

[L-2] Protocol giving wrong return, mislead the user.

Description: In `TSwapPool::swapExactInput` function, the function is intended to output amount sent to user after swap, but the `uint256 output` variable isn't used in the function and always return zero which misleads the user.

Impact: User sees that even after poolToken amount is sent to pool the output weth is zero which mislead the user.

Recommended Mitigation: Do following changes in `TSwapPool::swapExactInput` function:

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

-        uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount, inputReserves, outp
+        uint256 output = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputRese

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

-        _swap(inputToken, inputAmount, outputToken, outputAmount);
+        _swap(inputToken, inputAmount, outputToken, output);```
```

GAS

[G-1] lacking zero address check, unneccessary gas used if zero address enters.

Description: In `TSwapPool::constructor`,

Instances:

- `    i_wethToken = IERC20(wethToken);`
- `    i_poolToken = IERC20(poolToken);`
- In `PoolFactory::constructor`
- `    i_wethToken = wethToken;`

Impact: Unnecessary gas used if zero address try to enters

Recommended Mitigation: Add following lines to constructor:

- Here TokenAddress is i_wethToken || i_poolToken.

```
+ if(TokenAddress == address(0)){
+     revert();
+}
```

[G-2] Variable not used, take unneccessary gas.

Description: In `TSwapPool::deposit`, `poolTokenReserves` is not used but initialized and read from storage

Impact: Take unneccessary gas

Recommended Mitigation: Remove following line:

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

INFORMATIONALS

[I-1] Events not indexed, even if more than three parameters

Description: Swap event have more than 3 parameters and still not indexed it confuse anyone who take the trasaction logs off-chain.

Impact: It is very hard to see them Off-chain if not indexed.

Recommended Mitigation: Make those event indexed.

[I-2] In `PoolFactory::createPool` use .symbol() instead of .name for `liquidityTokenSymbol` variable.

Description: .name() is used at `liquidityTokenSymbol` function instead of .symbol

Impact: Takes more space to assign as .name()

Recommended Mitigation: Change according to following:

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