TSwap Audit Report

Version 1.0

toshiiki

May 9, 2024

TSwap Audit Report

toshiiki

May 9, 2024

Prepared by: toshiiki Lead Auditors: - toshiiki

Table of Contents

Protocol Summary

Disclaimer

toshiiki 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 toshiiki 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

Scope

Roles

Executive Summary

Issues found

Severity	# of Issues
High	4
Medium	2
Low	5
Info	4
Gas	0
:———-:	:————-:
Total	15

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resultion in lost fees

Description: the `getInputAmountBasedOnOutput` function is inteded to calculate the amount of tokens auser should deposit given an aount of tokens of output tokens. However, the function currently miscalculates the resulting amount. When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: Protocol takes more fees than expected from users

Proof of Concept:

Recommended Mitigation:

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

[H-2] Lack of Slippage protection in `TSwapPool::sawapExactOutput` causes users to potentially receive way fewer tokens

Description: The `swapExactOutput` function does not include any sort of slippage protection. this function is similar to what is done in `TSwapPool::swapExactInput` where the function specifies a `minOutputAmount`, the `swapExactOutpuut` function should specify a `maxInputAmount`.

Impact: If market condtitions lchange before the transaction processes, the user could get a much worse swap.

Proof of Concept: 1. The price of 1 WETH right now is 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = whatever 3. The function does not offer a maxInput amount 4. As the transaction is pending in the mempool, the market changes! And the price moves HUGE -> 1 WETH is now 10,000 USDC. 10x more than the user expected 5. The transaction completes, but the user sent the protocol 10,000 USDC instead of the expected 1,000 USDC

PoC

Recommended Mitigation: We should include a `maxInputAmount` so the user only has to spend up to a specific amount, and can predict how much they will spend on the protocol.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(outputAmount,
           inputReserves, outputReserves);
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to aloow users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they are willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculates the wapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas the `swapExactInput` funciton is the one that should be called because users specify the exact amount of input tokens, not output.

Impact: Users will swap the wrong awmont of tokens, which is a severe disruption of protocol functionality.

Proof of Concept:

PoC

Recommended Mitigation: consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function it accept a new parameter (i.e., `minWethToReceive` to be passed to `swapExactInput`)

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

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline. (MEV later)

[H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`

Description: The protocol follows a strict invariant of `x * y = k`, where: - `x` = the balance of the pool token - `y` = the balance of WETH - `k` = the constant product of the two balances

This means whenever the balnces 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 protocaol funds will be drained.

The following block of code is responsible for the issue:

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

Impact: A user could maliciously drain the protocol of funds by doing a lot of swawps and collecting the incentive given out by the protocol. Most simply put, the protocol's core invariant is broken.

Proof of Concept: 1. a user swaps 10 times and collects the extra incentive of `1_000_000_000_000_000_000` tokens 2. That user continues to swap until all the protocol funds are drained

PoC

Place the following test into `TSwapPool.t.sol`

```
    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 incentive mechanism. If you want to keep this in you shjoud account for the change in the x * y = k protocol invariant. Or, we should set aside tokens in the same way we do with fees.

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

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing transact8ions to complete even after the deadline has passed

Description: The `deposit` function accepts a deadline parameter, which according to the documentation is "The deadline for the transaction to be completed by". However, this parameter is never used. As a consequence, operations that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact: Transactions could be sent when market conditions are unfavorable to deposit even when adding a deadline parameter.

Proof of Concept: the `deadline` parameter is unused in the function

Recommended Mitigation:

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

[M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant

Description: many non-standard ERC20 tokens have additional incentuives or fees that can cause the amount of tokens in a transaction to vary. This causes the protocol invariant of `x * y = k` to be broken frequently.

Impact: the invariant of the protocol is broken, leading to inconsistent operation and possible manipulation of the protocol

Proof of Concept:

Recommended Mitigation:

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing events to emit incorrect information

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, whereas the `wethToDeposit` value should go second.

Impact: Event mission incorrect, leading to off-chain functions potentially malfunctioning.

Recommended Mitigation:

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

[L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given

Description: the `swapExactInput` function is expected to return the actual amount of tokens bought by the caller. However, while it declares the named return value `output` it is never assigned a value, nor uses and explicit return statement.

Impact: The return value will always be 0, giving incorrect information to the caller.

Recommended Mitigation:

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

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

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

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

[L-3] PUSH0 is not supported by all chains

Description: 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.

- Found in src/PoolFactory.sol Line: 15

  ```solidity
  pragma solidity 0.8.20;
  ```

- Found in src/TSwapPool.sol Line: 15

  ```solidity
  pragma solidity 0.8.20;
  ```

[L-4] `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.

- Found in src/TSwapPool.sol Line: 305

  ```solidity
      function swapExactInput(
  ```

[L-5] 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.

- Found in src/TSwapPool.sol Line: 276

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

- Found in src/TSwapPool.sol Line: 302

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

- Found in src/TSwapPool.sol Line: 463

```
                    1e18,
```

- Found in src/TSwapPool.sol Line: 472

```
                    1e18,
```

Informationals

[I-1] Error `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed

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

[I-2] Lacking Zero address checks

```
constructor(address wethToken) {
+        if(wethToken == address(0)) {
+                revert();
+        }
         i_wethToken = wethToken;
    }
```

[I-3] `PoolFactory::liquidityTokenSymbol` should use `.symbol()` instead of `.name()`

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

[I-4] Events are missing indexed fields

Description: Indexed event fields make fields 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 ot index the maximum allowed per event (three fields in this case). 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.

Gas