title	author	date
TSwap Audit Report	Franklyn Ezeugonna	March 3, 2024

TSwap Audit Report

Prepared by: Franklyn Ezeugonna Lead Auditors:

- Franklyn Ezeugonna

Assisting Auditors:

- None

Table of Contents

About me

I'm passionate about uncovering vulnerabilities in systems and smart contract , always curious and eager to learn . Most importantly, I love making new friends . Feel free to reach out.

Disclaimer

I Franklyn Ezeugonna makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the 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

Audit Details

The findings described in this document correspond the following commit hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

```
./src/
-- PuppyRaffle.sol

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

Protocol Summary

This project is meant to be a permissionless way for users to swap assets between each other at a fair price. You can think of T-Swap as a decentralized asset/token exchange (DEX). T-Swap is known as an Automated Market Maker (AMM).

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

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	9
Total	17

high

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

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount 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: my challeng is to write a POC for this.

Recommended Mitigation:

```
function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
```

```
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
-           ((inputReserves * outputAmount) * 10000) / ((outputReserves -
outputAmount) *    997);
+           ((inputReserves * outputAmount) * 1000) / ((outputReserves -
outputAmount) * 997);
    }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` 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 `swapExactOutput` function should specify a `maxInputAmount`.

Impact: If market conditions change 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 the input parameters will be the following
    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 to -> 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 insted of the expected 1,000 USDC

my challeng is to write a POC for this.

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,
        IERC20 outputToken,
        uint256 outputAmount,
+       uint256 maxInputAmount,
        uint64 deadline
    )
```

```
.
.
.
   inputAmount = getInputAmountBasedOnOutput( outputAmount, inputReserves,
outputReserves );
+  if (inputputAmount < maxOutputAmount) {
+           revert();
        }
```

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

Description: The `TSwap::sellPoolTokens` function is intended to allow users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell in the `poolTokenAmount` parameter. However the function currently miscalculats the swapped amount.

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

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

Proof of Concept: my challeng is to write a POC for this.

Recommended Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`, Note that this would also require changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to 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));
+           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.

[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 the weth token
- `K`; The constant product of the two balances

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 slowly over time the protocol 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 swaps and collecting the extra incentive given out by the protocol.

The protocol 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.

▶ Proof Of Code

Place the following 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 = 1e5;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        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, we should 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 transaction to complete even after the deadline

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 cosequence , operations that add liquidity to the pool might be executed at unexpected times. In market conditions where the deposit rate is unfavourable.

Impact: Transaction could be sent when market conditions are unfavourable to deposit, even when adding a deadline parameter.

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

Recommended Mitigation: Consider making the following change to the function.

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

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

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

Impact: Event emmision is incorrect , leading in off-chain functions potentially malfunctioning.

Proof of Concept: my challeng is to write a POC for this.

Recommended Mitigation:

```
- 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 an explicit return statement.

Impact: Thr return value will always be 0 , giving incorrect infornation to the caller.

Proof of Concept: write a poc , to show that no matter what you do , you will always a return a zero

Recommended Mitigation:

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

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

```
+        output = getOutputAmountBasedOnInput( inputAmount,inputReserves,
outputReserves );


-        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);
    }
```

Informationals

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

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

[I-2] Lacking zero address check in the following

found in `PoolFactory::constructor`

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

found in `TSwapPool::constructor`

```
 constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+     if (wethToken == address(0) || poolToken == address(0)) {
+           revert("Invalid address");
+       }
      i_wethToken = IERC20(wethToken);
```

```
        i_poolToken = IERC20(poolToken);
    }
```

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

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

[I-3] 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 `PoolFactory.sol` [Line: 35]

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

- Found in `TSwapPool.sol` [Line: 52]

  ```
      event LiquidityAdded(
  ```

- Found in `TSwapPool.sol` [Line: 57]

  ```
      event LiquidityRemoved(
  ```

- Found in `TSwapPool.sol` [Line: 62]

  ```
      event Swap(
  ```