Puppy Raffle Audit Report

Prepared by Pluton

Version 1.0

Lead Auditor

Herman Effendi

August 12, 2024

Table of Contents

Introduction

The Pluton team strives to identify as many vulnerabilities as possible within the allotted time but accepts no responsibility for the results detailed in this document. Their security audit should not be considered an endorsement of the business or product. The audit was limited in scope and focused exclusively on the security aspects of the Solidity code for the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

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

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

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

Roles

- Liquidity provider : Someone who act give a many tokens for the pool
- Users : Someone who use the functionality the protocol for swapping tokens

Executive Summary

Issues found

Severity	Number of issues found
High	3
Medium	2
Low	0
Info	3
Total	8

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput`, causes the protocol take fee too much from user.

Description

The `getInputAmountBasedOnOutput` is intended to calculate the amount of tokens a users should deposit give an amount of token of output token. However, the function currently miscalculate. it scale he amount by 10_000 instead of 1_000.

Impact

The protocol takes fees too much rather than expected.

Proof of Concepts

Code

```
function testGetInputAmountBasedOnOutput() public view {
        uint256 outputAmount = 50e18;
        uint256 inputReserves = 200e18;
        uint256 outputReserves = 100e18;

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

```
7           uint256 result = pool.getInputAmountBasedOnOutput(outputAmount,
                inputReserves, outputReserves);
8
9           assert(result >= expected);
10      }
```

Recommended mitigation

```
1       function getInputAmountBasedOnOutput(
2           uint256 outputAmount,
3           uint256 inputReserves,
4           uint256 outputReserves
5       )
6           public
7           pure
8           revertIfZero(outputAmount)
9           revertIfZero(outputReserves)
10          returns (uint256 inputAmount)
11      {
12  -       return ((inputReserves * outputAmount) * 10000) / ((outputReserves
            - outputAmount) * 997);
13  +       return ((inputReserves * outputAmount) * 1000) / ((outputReserves
            - outputAmount) * 997);
14      }
```

[H-2] `TSwapPool::sellPoolTokens` have mismatche input and output token, causing user to receive incorrect amount of token

Description

The `sellPoolTokens` function is intended o allow users to easily ell pool tokens and receive eth in exchange. Users indicate how many pool tokens they are willing to sell as a parameter. However, the function currently miscalculates the swaped amount.

This due to the fact that he `swapExactOutput` function is called, whereas `swapExactInput` function is the one that should be called because the user specify the input amount, not the output amount.

Impact

Users will swap the wrong amount of token. which is a severe distruption of protocol functionality.

Recommended mitigation

Consider change the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require `minWethToReceive` to be pass in `swapExactInput`.

```
1       function sellPoolTokens(
```

```
2           uint256 poolTokenAmount
3   +       uint256 minWethToReceive,
4           ) external returns (uint256 wethAmount) {
5   -       return swapExactOutput(i_poolToken, i_wethToken,
        poolTokenAmount, uint64(block.timestamp));
6   +       return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken
        , minWethToReceive, uint64(block.timestamp));
7       }
```

[H-3] In `TSwapPool::_swap` the extra tokens given to users to after every `swapCount` breaks the protocol invariant of `x * y = k`

Description

The protocol follows strict invariant of `x * y = k`. where : `x` : The balance of pool token `y` : The balance of eth `k` : The constant product formula, the ratio beetween

This means, that whenever the balances changes in the protocol, the ratio between two amount remain constant, hence the `k`. However, this is broken due to extra incentive in the `_swap` function. Meaning that overtime the protocol funds will drained.

The following th below code

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

Impact

A user could maliciuosly drain the protocol of funds by doing alot of swap and collecting extra incentive given out ot protocol.

Proof of Concepts

Recommended mitigation

Remove the extra incentive if the protocol want to keep the balance or we should set aside tokens in the same way we do this in fees.

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

Medium

[M-1] `TSwapPool::deposit` is missing deadline check, causing the transaction to complete even after the deadline

Description

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

Impact

Transactions could be sent when the market conditions are unfavorable to deposit even we adding deadline as a parameter.

Proof of Concepts

The `deadline` parameter is unused

Recommended mitigation

Consider making the following function change

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

[M-2] Lack of slippage protection in `TSwapPool::SwapExactOutput` causes users to potentially receive way fewer token.

Description

The `SwapExactOutput` function doesn't include any slippage protection. This function is similar to what is done im `SwapExactInput`, where the function specifies `minOutputAmount`. `SwapExactInput` should specifies a `maxInputAmount`.

Impact

If the market changes suddenly, it could lead to users experiencing less favorable swap outcomes.

Proof of Concepts

1. The price of 1 WETH right now is 1_000 USDC.
2. User inputs a `SwapExactOutput` looking for 1 WETH
    - inputToken USDC
    - outputToken WETH
    - outputAmount 1
    - deadline whatever
3. The function doesn't offer `maxInputAmount`
4. as the transaction pending in the mempool, the market changes!! and the price maybe around 1 WETH -> 10_000 USDC, 10x more than user expected.
5. The transaction completes, but the user sent the protocol 10_000 instead of 1_000 USDC.

Recommended mitigation

We should include `maxInputAmount` so the user only has spend up to a specify amount as well as predict how much they will spend in protocol.

```
 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, outputReserves);
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
    }
```

Low

[L-1] `TSwapPool::LiquidityAdd` event has a parameter out of order

Description

When the `LiquidityAdded` event is emitted by `TSwapPool::_addLiquidityMintAndTransfer` function, it's logs values in an incorrect order. The `PoolTokenDeposit` value should go in the third parameter position, whereas the `wethToDeposit` value should go in second.

Impact

Event emitted is incorrect, may lead to incorect filling parameter as well

Recommended mitigation

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

[L-2] Default value returned by `SwapExactInput` result in incorect return value given

Description

The `SwapExactInput` function is expected to return th actual amount of token bought by caller. However, while it declares the named return value `output` it is never assigned by value, nor uses explicit return statement.

Impact

The return value is always zero, it always give incorrect information for the caller.

Proof of Concepts

Code

```
function testSwapExactInput() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 10e18);
        poolToken.approve(address(pool), 10e18);
        pool.deposit(10e18, 0, 10e18, uint64(block.timestamp));
        vm.stopPrank();


        vm.startPrank(user);
        weth.approve(address(pool), 1e18);
```

```
11          uint256 result = pool.swapExactInput(weth, 1e18, poolToken, 1
                e17, uint64(block.timestamp));
12          vm.stopPrank();
13
14          assert(result == 0);
15      }
```

Recommended mitigation

Shoul be corrected the name variable as result

```
1       function swapExactInput(){
2           ...
3   -       returns (uint256 output)
4   +       returns (uint256 outputAmount)
5           ...
6           }
```

Informational

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

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

[I-2] Lacking zero address

```
1       constructor(address wethToken) {
2   +       if(weth == address(0)){
3   +           revert();
4   +       }
5           i_wethToken = wethToken;
6       }
```

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

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