PuppyRaffle Audit Report

Version 1.0

skipper audits

October 2, 2024

Protocol Audit Report

skipper

oct 2,2024

Prepared by: [Skipper] Lead Auditors: - Skipper

Table of Contents

  - [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after deadline
  - [M-2] Rebase ,fee-on-transfer and ERC777 tokens break protocol invariant.

- Low

  - [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
  - [L-3] Unused return varable in `TSwapPool::swapExactInput` function and Provide Natspec for the function

- Informational

  - [I-1] PUSH0 is not supported by all chains(0.8.20 version)
  - [I-2] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  - [I-3] lacks a zero address check
  - [I-4] `PoolFactory::createPool(address tokenAddress)` function shoul use `.symbol()` instead of `.name()`.
  - [I-5]`PoolFactory::Event is missing indexed fields`
  - [I-6] lacks a zero address check
  - [I-6]Unused variable `poolTokenReserves` in `PoolFactory::deposit`function
  - [I-7]Better to follow CEI in `poolTokenReserves` in `PoolFactory::deposit` function
  - [I-8] Define and use constant variables instead of using literals

- Gas

  - [G-1] The `TSwapPool::swapExactInput` function should be External if not used internally

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) because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset. It is similar to Uniswap. To understand Uniswap, please watch this video: Uniswap Explained

Disclaimer

Skipper 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
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

- 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 ## Scope
- In Scope:

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

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	4
Medium	2
Low	3
info	8
gas	1
Total	18

Findings

High

[H-1] logic/typo error in `TSwapPool::getInputAmountBasedOnOutput` function

Description:

In the `TSwapPool::getInputAmountBasedOnOutput` function `((inputReserves * outputAmount)* 10000)` is multiplied by 10_000 instead of 1_000 seriously affecting the protocols functionality.

Impact Protocol takes more fees than expected from users.

Proof of Concept:

run this test in the `PoolFactoryTest.t.sol`and check the logs

Proof Of Code

```
function testHighFees() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        uint256 PoolTokeninputReserve = poolToken.balanceOf(address(pool));
        uint256 wethReserve = weth.balanceOf(address(pool));
        uint256 outputAmount = 2 ether;
        uint256 actualInputAmount = pool.getInputAmountBasedOnOutput(outputAmount, PoolTokeninputReserve, wethReserve);
        console.log("actualInputAmount", actualInputAmount);
         vm.stopPrank();
        // but we get 20.469 ETH which is 10X what we need.
   }
```

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        //@audit magic numbers
        // 10,000 instead of 1,000 High
-         return ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997);

+       return ((inputReserves * outputAmount) * 1_000) / ((outputReserves - outputAmount) * 997);
    }
```

additionaly, use constant variable instead of using iterals eg:-

```
uint256 constant PRECISION_1=1000;
uint256 constant PRECISION_2=997;

 return ((inputReserves * outputAmount) * PRECISON_1) / ((outputReserves - outputAmount) * PRECISION_2);
```

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

Description:

The `TSwapPool::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 1000 USDC. 2. user inputs a `swapExactOutput` looking for 1 weth. 1. inputToken=USDC 2. outputToken=weth 3. outputAmount=1 4. deadline= as the user needs 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 eg:- 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.

Recommended Mitigation:

```
        function swapExactOutput(
            IERC20 inputToken,
            IERC20 outputToken,
            uint256 outputAmount,
            uint64 deadline,
+           uint256 maxInputAmount
        )
            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);
        }
```

[H-3] The `swapExactOutput` in the `TSwapPool::sellPoolTokens` function takes parameters in the Wrong order causing the users to receive the incorrect amount of tokens.

Description:

The `TSwapPool::sellPoolTokens` function is intended to allow 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 swap amount.

This is due to the fact that the `SwapExactOutput` function is called, whereas the `SwapExactinput` function is the one that supposed to be called , Because users specify the exact amount of input tokens,not output

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

Recommended Mitigation:

Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note thet this would also require changing the `sellPoolTokens` function to accept a new parameter(ie, `minWethToReceive`) to be passeed 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 must be wise to add a deadline to the function,as there is currently no deadline.

[H-4] In the `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, 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.

The follow 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.

Proof of Concept: place the folloeing test in the `TSwapPool.t.sol`

Poc

```
    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 = 1e18;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 1000e18);
        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++;
-           // Fee-on-transfer
-           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 transactions to complete even after deadline

Description: The `TSwapPool::deposit` accepts a deadline parameter " @param deadline The deadline for the transaction to be completed by". However ,this parameter is not used.As a consequence, operations that add liquidity to the pool might be executed times, in market conditions where the deposit rate is unfavourable .

Impact: The transactions could be sent when the market conditions are unfavourable to deosit,even when adding a deadline parameter.

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

Recommended Mitigation:

```
    function deposit(
            uint256 wethToDeposit,
            uint256 minimumLiquidityTokensToMint,
            uint256 maximumPoolTokensToDeposit,
            // @audit High deadline completely ignored
            uint64 deadline
        )
            external
            revertIfZero(wethToDeposit)
```

```
+        revertIfDeadlinePassed(deadline)
         returns (uint256 liquidityTokensToMint)
```

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

Description: Some tokens take a transfer fee (e.g. STA, PAXG), some do not currently charge a fee but may do so in the future (e.g. USDT, USDC).

Some tokens do not return a bool (e.g. USDT, BNB, OMG) on ERC20 methods.

Some tokens allow reentrant calls on transfer (e.g. ERC777 tokens).

see more info : https://github.com/d-xo/weird-erc20?tab=readme-ov-file

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order

Description:

when the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` function ,it logs values in incorrect order. The `poolTokensToDeposit` value should go in the 3rd parameter position and `wethToDeposit` should go in the 2nd parameter position.

Impact: Event emission is incorrect ,leading to off-chain functionality potentially malfunctioning.

Recommended Mitigation:

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

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

[L-3] Unused return varable in `TSwapPool::swapExactInput` function and Provide Natspec for the function

Description:

The `TSwapPool::swapExactInput` function does not return the variable that is declared in the function causing the return to be always 0.

Impact:

causes function to return 0 always giving wrong information.

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(outputAmount, minOutputAmount);
        }

-        _swap(inputToken, inputAmount, outputToken, outputAmount);

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

Informational

[I-1] PUSH0 is not supported by all chains(0.8.20 version)

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.

[I-2] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed

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

[I-3] lacks a zero address check

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

[I-4] `PoolFactory::createPool(address tokenAddress)` function shoul use `.symbol()` instead of `.name()`.

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

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

[I-5]`PoolFactory::Event is missing indexed fields`

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 43

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

- Found in src/TSwapPool.sol Line: 44

  ```
      event LiquidityRemoved(address indexed liquidityProvider, uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
  ```

- Found in src/TSwapPool.sol Line: 45

  ```
      event Swap(address indexed swapper, IERC20 tokenIn, uint256 amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
  ```

Description:

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.

[I-6] lacks a zero address check

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

[I-6]Unused variable `poolTokenReserves` in `PoolFactory::deposit`function

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

[I-7]Better to follow CEI in `poolTokenReserves` in `PoolFactory::deposit`function

```
+    liquidityTokensToMint = wethToDeposit;
     _addLiquidityMintAndTransfer(wethToDeposit, maximumPoolTokensToDeposit, wethToDeposit);
-    liquidityTokensToMint = wethToDeposit;
```

[I-8] 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: 231

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

- Found in src/TSwapPool.sol Line: 248

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

- Found in src/TSwapPool.sol Line: 374

```
            1e18, i_wethToken.balanceOf(address(this)), i_poolToken.balanceOf(address(this))
```

- Found in src/TSwapPool.sol Line: 380

```
            1e18, i_poolToken.balanceOf(address(this)), i_wethToken.balanceOf(address(this))
```

Gas

[G-1] The `TSwapPool::swapExactInput` function should be External if not used internally

```
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
-         public
+         external
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    //@audit info unused return variable
    {

    }
```