PuppyRaffle Audit Report

Version 1.0

bumble.c

July 21, 2024

Protocol Audit Report

bumble.c

July 21, 2024

Prepared by: Bumble Lead Security Researcher:

- Medium
  * [M-1] `TSwapPool::deposit` receives a deadline but does not use it, allowing deposits to happen after the deadline provided
- Low
  * [L-1] `TSwapPool::_addLiquidityMintAndTransfer` event emission args are in the wrong order, leading to confusion of the contract's behavior
  * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informational
  * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error is not being used
  * [I-2] Lacking zero address check on `PoolFactory::constructor`
  * [I-3] `PoolFactory::createPool` liquidityTokenSymbol should concat with the token symbol
  * [I-4] `In TSwapPool::deposit`, if the deposit if less than the minimum deposit, it reverts with a constant `MINIMUM_WETH_LIQUIDITY`, which is not informative and will repeat the same error message for all deposits
  * [I-5] `TSwapPool::deposit` is not using `poolTokenReserves`, wasting gas and not using the variable
  * [I-6] `TSwapPool::getOutputAmountBasedOnInput` is using magic numbers, which makes the code harder to understand
  * [I-7] `TSwapPool::getInputAmountBasedOnOutput` is using magic numbers, which makes the code harder to understand
  * [I-8] `TSwapPool:swapExactInput` does not have natspec comments, making it harder to understand the function's purpose on the protocol
  * [I-9] `In TSwapPool::deposit`, if the deposit if less than the minimum deposit, it reverts with a constant `MINIMUM_WETH_LIQUIDITY`, which is not informative and will repeat the same error message for all deposits
  * [I-10] `TSwapPool::deposit` is not using `poolTokenReserves`, wasting gas and not using the variable

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

The Bumble team 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

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

Audit Details

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda ## Scope:

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

Compatibilities

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum

- Tokens:
  - Any ERC20 token

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

The audit was conducted on the `TSwap` protocol. It took the team 3 days to complete the audit, and the team found 16 issues in the protocol. The issues found were of varying severity, with 4 high, 1 medium, 2 low, and 10 informational issues found. The team recommends that the protocol follows the CEI (Check-Effect-Interact) pattern, and that the protocol uses constants to make the code more readable and easier to understand. The team also recommends that the protocol uses a deadline to ensure that transactions are completed before the deadline provided. The team also recommends that the protocol removes the extra incentive mechanism in the `_swap` function, as it breaks the protocol's core invariant. The team also recommends that the protocol uses a `maxInputAmount` parameter to protect users from slippage. The team also recommends that the protocol changes the implementation of the `sellPoolTokens` function to use `swapExactInput` instead of `swapExactOutput` . The team also recommends that the protocol changes the order of the arguments in the event `LiquidityAdded` to match the order of the arguments in the event. The team also recommends that the protocol assigns the correct value to the `output` variable and returns it at the end of the function. The team also recommends that the protocol adds a check to ensure that the transaction is completed before the deadline provided. This was a great project to work on, and the team is looking forward to working on more projects in the future.

Issues found

Severity	Numbers of Issues found
High	3
Medium	1
Low	2

Severity	Numbers of Issues found
Info	10
Total	16

Findings

High

[H-1] Incorrect fee calculation at `TSwapPool::getInputAmountBasedOnOutput` causing the protocol to take more fees than it should

Description

The `TSwapPool::getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens an user should deposit to receive a certain amount of tokens. However, the function miscalculates the resulting amount. When calculating the fees, it scales the amount by 10_000 instead of 1_000, causing the fees to be 10 times higher than they should be.

Impact The users will be charged way too much, losing money and dont using the protocol again

Recommended mitigation Easiest way would be to remove the extra zero from the calculation, making the fees 0.3% instead of 3%.

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

However its good to know that using constant numbers should help to make the code more readable and easier to understand, also it could prevent errors like this one by naming a number. Use a design like this to avoid this kind of error:

```
+    uint256 constant TAX_FEE = 997;
+    uint256 constant FEE_DENOMINATOR = 1000;
+    return
+        ((inputReserves * outputAmount) * FEE_DENOMINATOR) /
+        ((outputReserves - outputAmount) * TAX_FEE);
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput`, causes users to receive less tokens than expected

Description The function `TSwapPool::swapExactOutput` does not have any slippage protection, which means that the user may receive less tokens than expected. This can happen when the price of the token changes between the time the transaction is sent and the time it is executed. This function is similar to what is done in `TSwapPool::swapExactInput`, where the function specifies `minOutputAmount`, the `swapExactOutput` function should also have a `maxInputAmount` parameter to protect the user from slippage.

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 1. inputToken = USDC 2. outputToken = 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 10000 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

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 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 miscalculaes the swapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas 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 protcol functionality.

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 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.

[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, 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.

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 untill 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 = 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, 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.

```diff
-         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` receives a deadline but does not use it, allowing deposits to happen after the deadline provided

Description The `TSwapPool::deposit` function receives a `deadline` parameter, which according to the documentation `The deadline for the transaction to be completed by`. However, this parameter is not used in the function. As a consequence, operations that add liquidity to the pool might never be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact The transaction may complete after the deadline provided, allowing transactions to be sent on conditions that are not favorable to the user.

Proof of Concepts

1. Call the `deposit` function with a deadline in the past.
2. The transaction will complete successfully, allowing the user to deposit tokens after the deadline provided.
3. The user should not be able to deposit tokens after the deadline provided.

Recommended mitigation Add a check to ensure that the transaction is completed before the deadline provided. Consider making the following change:

```
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        // @written- not being used, probably high
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
```

Low

[L-1] `TSwapPool::_addLiquidityMintAndTransfer` event emission args are in the wrong order, leading to confusion of the contract's behavior

Description At the internal function `_addLiquidityMintAndTransfer`, the event `Liquidity added` is emitted with the wrong order of arguments. The event should be emitted with the following order of arguments:

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

But instead is emmited with the following order:

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

Impact This issue does not have a direct impact on the contract's functionality. However, it may confuse users and developers who are trying to understand the contract's behavior, leading to off-chain malfunctions.

Proof of Concepts 1. Deploy the contract. 2. Call the `_addLiquidityMintAndTransfer` function. 3. Check the event `LiquidityAdded` and verify that the arguments are in the wrong order, pool tokens will appear as weth and weth as pool tokens.

Recommended mitigation Change the order of the arguments in the event `LiquidityAdded` to match the order of the arguments in the event.

```
-     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 `ouput` it is never assigned a value, nor uses an explict return statement

Impact The return value will always be 0, given incorrect information to the caller

Recommended mitigation Assign the correct value to the `output` variable and return it at the end of the function

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

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

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

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

Informational

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error is not being used

Description The `PoolFactory__PoolDoesNotExist` error is not being used in the `PoolFactory` contract. This error is defined in the `PoolFactory` contract, but it is not used in any of the functions.

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

[I-2] Lacking zero address check on `PoolFactory::constructor`

```
    constructor(address wethToken) {
+     require(wethToken != address(0), "WethToken must not be zero
    address");
+         i_wethToken = wethToken;
+     }
-         i_wethToken = wethToken;
```

[I-3] `PoolFactory::createPool` liquidityTokenSymbol should concat with the token symbol

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

[I-4] In `TSwapPool::deposit`, if the deposit if less than the minimum deposit, it reverts with a constant `MINIMUM_WETH_LIQUIDITY`, which is not informative and will repeat the same error message for all deposits

```
    revert TSwapPool__WethDepositAmountTooLow(
-         MINIMUM_WETH_LIQUIDITY,
        wethToDeposit
    );
```

[I-5] `TSwapPool::deposit` is not using `poolTokenReserves`, wasting gas and not using the variable

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

[I-6] `TSwapPool::getOutputAmountBasedOnInput` is using magic numbers, which makes the code harder to understand

```
-     uint256 inputAmountMinusFee = inputAmount * 997;
-     uint256 numerator = inputAmountMinusFee * outputReserves;
-     uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee
    ;

+     uint256 constant TAX_FEE = 997;
+     uint256 constant FEE_DENOMINATOR = 1000;
+     uint256 inputAmountMinusFee = inputAmount * TAX_FEE;
+     uint256 numerator = inputAmountMinusFee * outputReserves;
+     uint256 denominator = (inputReserves * FEE_DENOMINATOR) +
    inputAmountMinusFee;
```

[I-7] `TSwapPool::getInputAmountBasedOnOutput` is using magic numbers, which makes the code harder to understand

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

+     uint256 constant TAX_FEE = 997;
+     uint256 constant FEE_DENOMINATOR = 1000;
+     return
+         ((inputReserves * outputAmount) * FEE_DENOMINATOR) /
+         ((outputReserves - outputAmount) * TAX_FEE);
```

[I-8] `TSwapPool:swapExactInput` does not have natspec comments, making it harder to understand the function's purpose on the protocol

```
+     /// @notice Swaps an exact amount of input tokens for as much
    output tokens as possible
+     /// @param inputToken The token to be swapped
+     /// @param inputAmount The amount of input tokens to be swapped
+     /// @param outputToken The token to receive
+     /// @param minOutputAmount The minimum amount of output tokens to
    receive
+     /// @param deadline The deadline for the swap to be executed
  function swapExactInput(
         IERC20 inputToken,
         uint256 inputAmount,
         IERC20 outputToken,
         uint256 minOutputAmount,
         uint64 deadline
     )
```

[I-9] In `TSwapPool::deposit`, if the deposit if less than the minimum deposit, it reverts with a constant `MINIMUM_WETH_LIQUIDITY`, which is not informative and will repeat the same error message for all deposits

```
      revert TSwapPool__WethDepositAmountTooLow(
-         MINIMUM_WETH_LIQUIDITY,
            wethToDeposit
        );
```

[I-10] `TSwapPool::deposit` is not using `poolTokenReserves`, wasting gas and not using the variable

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