Protocol Audit Report

Version 1.0

Cyfrin.io

December 26, 2025

T-Swap Audit Report

Dengi-XCVI

March 7, 2023

Prepared by: Dengi Lead Auditors: - dengi

Table of Contents

- Medium
  - [M-1] `TSwapPool::deposit` function doesn't use the `uint64 deadline` parameter
- Low
  - [L-1] Wrong order of parameters when emitting event `TSwapPool::LiquidityAdded`
  - [L-2] `TSwapPool::swapExactInput` is returning unused `uint256 output`
- Informationals
  - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExis` error is not used
  - [I-2] Lacking zero address checks in `PoolFactory::constructor`
  - [I-3] `PoolFactory::createPool` should use `.symbol()` instead of using `.name()` twice

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 DENGI 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
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L

	Impact			
	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 to the following commit hash: Commit Hash

Scope

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

Roles

- Liquidity Providers: Users that provide liquidity to the pools in exchange of a claim on the pool's fees.
- Swapper: Any users that swaps tokens in the pool

Executive Summary

The audit has been conducted using a combination of manual review, static analysis tools (Slither, Aderyn) and fuzz testing (using foundry's built in fuzzer).

Issues found

Severity	Issues Found
High	4
Medium	1
Low	2
Info,Gas	3
Total	10

Findings

High

[H-1] `TSwapPool::getInputAmountBasedOnOutput` has a hardcoded value of `10_000` to multiply the numerator

Description: The documentation of the project states that numbers `997` and `1000` should be used to calculate the fee swaps. But the function `TSwapPool::getInputAmountBasedOnOutput` is using `10000`.

Impact: High, will result in higher fees when swapping.

Proof of Concept:

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

Recommended Mitigation: Change the number `10000` to `1000` in the code highlighted above. It is also best practice to use named constant variables to avoid having so called "magic numbers" in the codebase and prevent similar errors.

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput`

Description: `TSwapPool::swapExactOutput` has no slippage protection. This function is similar to what is done in `TSwapPool::swapExactInput`, which in contrast to the function subject to vulnerability, implements checks to avoid slippage. In particular the `swapExactInput` function uses the `minOutputAmount` variable to avoid unintended consequences.

Impact: High, this can result in users paying an unexpected amount for the swap due to changing market conditions/pool balances

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

Recommended Mitigation:

Add the variable `uint256 maxInputAmount` to the function arguments and use it to check for unexpected price changes. Add also a custom `error TSwapPool__InputAmountTooHigh(uint256 inputAmount, uint256 maxInputAmount);`

```
    inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );
+   if (inputAmount > maxInputAmount) {
+       revert TSwapPool__InputAmountTooHigh(inputAmount, maxInputAmount);
}
```

[H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing the user to receive the wrong amounts

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,
```

```
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       }
```

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, 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.

```
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 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`.

```
1       function testInvariantBroken() public {
2           vm.startPrank(liquidityProvider);
3           weth.approve(address(pool), 100e18);
4           poolToken.approve(address(pool), 100e18);
5           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
6           vm.stopPrank();
```

```
7
8           uint256 outputWeth = 1e17;
9
10          vm.startPrank(user);
11          poolToken.approve(address(pool), type(uint256).max);
12          poolToken.mint(user, 100e18);
13          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
14          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
15          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
16          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
17          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
18          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
19          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
20          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
21          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
22
23          int256 startingY = int256(weth.balanceOf(address(pool)));
24          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25
26          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31          assertEq(actualDeltaY, expectedDeltaY);
32      }
```

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.

```
1 -         swap_count++;
2 -         // Fee-on-transfer
3 -         if (swap_count >= SWAP_COUNT_MAX) {
4 -             swap_count = 0;
5 -             outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
6 -         }
```

Medium

[M-1] `TSwapPool::deposit` function doesn't use the `uint64 deadline` parameter

Description: The `TSwapPool::deposit` function takes a parameter `deadline` which is meant to be a deadline after which the deposit operation is not valid anymore. The parameter however is left unused, this might cause the transaction to go through later causing confusion or worse deposit conditions for the user. For example, a user might think that his deposit failed and try to deposit again.

Impact: Medium, missing checks for `deadline` might lead to bad user experience and confusion.

Proof of Concept: 1. A user might call `deposit` function and send funds to the pool. 2. The transaction might take longer than expected to go through but it won't revert or communicate anything to the user. 3. The deposit might happen later than intended.

Recommended Mitigation: Add a the `revertIfDeadlinePassed(deadline)` modifier to the function to the `TSwapPool::deposit` function.

Low

[L-1] Wrong order of parameters when emitting event `TSwapPool::LiquidityAdded`

Description: The parameters when the event is emitted in the `TSwapPool:_addLiquidityMintAndTransfer` function are out of order

Impact: Low, the event emitted is incorrect

Recommended Mitigation:

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

[L-2] `TSwapPool::swapExactInput` is returning unused `uint256 output`

Description: In the function declaration it is stated that the function returns `uint256 output`, however the variable is never declared or used, so the function will always return 0.

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

Recommended Mitigation: Eliminate the return statement in the function decalaration if you don't intend to use the variable. Otherwise modify the function as you see fit.

Informationals

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

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

[I-2] Lacking zero address checks in `PoolFactory::constructor`

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

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

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