Protocol Audit Report

Version 1.0

M3dython

October 14, 2024

PuppyRaffle Audit Report

M3dython

October 10, 2024

Prepared by: M3dython Lead Security Researcher:

- Low
  - [L-1]: `public` functions not used internally could be marked `external`
  - [L-2]: Define and use `constant` variables instead of using literals
  - [L-3]: Event is missing `indexed` fields
  - [L-4]: PUSH0 is not supported by all chains
  - L-5: Large literal values multiples of 10000 can be replaced with scientific notation
  - L[-6]: Unused Custom Error
  - [L-7] `TSwapPool::LiquidityAdded` event has parameters out of order
  - [L-8] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informational
  - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  - [I-2] Lacking zero address checks
  - [I-3] `PoolFacotry::createPool` should use `.symbol()` instead of `.name()`
  - [I-4] Event is missing `indexed` fields

Audit Details

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

```
22bbbb2c47f3f2b78c1b134590baf41383fd354f
```

Scope

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

Protocol Summary

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

Core Invariant

Our system works because the ratio of Token A & WETH will always stay the same. Well, for the most part. Since we add fees, our invariant technially increases.

```
x * y = k
```

- x = Token Balance X
- y = Token Balance Y
- k = The constant ratio between X & Y

```
y = Token Balance Y
x = Token Balance X
x * y = k
x * y = (x + Δx) * (y -Δ y)Δ
x = Change of token balance XΔ
y = Change of token balance Yβ
 = Δ(y / y)α
 = Δ(x / x)

Final invariant equation without fees:Δ
x = ββ(/(1-)) * xΔ
y = αα(/(1+)) * y

Invariant with feesρ
 = fee (between 0 & 1, aka a percentage)γ
 = (1 - p) (pronounced gamma)Δ
x = ββ(/(1-)) * γ(1/) * xΔ
y = αγαγ(/1+) * y
```

Executive 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).

Issues found

Severity	Number of issues found
High	4
Medium	1

Severity	Number of issues found
Low	8
Info	2
Total	15

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.

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocll 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.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10_000) / ((
    outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1_000) / ((
    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 transaciton 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 = 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: 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.

Proof of Concept:

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));
```

```
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.

```
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));
```

```
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` is missing deadline check causing transactions 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 consequence, operationrs that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact: Transactions could be sent when market conditions are unfavorable 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, // LP tokens -> if empty,
            we can pick 100% (100% == 17 tokens)
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

Low

[L-1]: `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

1 Found Instances

- Found in src/TSwapPool.sol Line: 298

```
    function swapExactInput(
```

[L-2]: 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: 276

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

- Found in src/TSwapPool.sol Line: 295

```
            ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 454

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
                1e18,
```

[L-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 src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
    event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
    event Swap(
```

[L-4]: PUSH0 is not supported by all chains

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.

2 Found Instances

- Found in src/PoolFactory.sol Line: 15

```
pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
pragma solidity 0.8.20;
```

L-5: Large literal values multiples of 10000 can be replaced with scientific notation

Use `e` notation, for example: `1e18`, instead of its full numeric value.

3 Found Instances

- Found in src/TSwapPool.sol Line: 45

```
    uint256 private constant MINIMUM_WETH_LIQUIDITY = 1
        _000_000_000;
```

- Found in src/TSwapPool.sol Line: 294

```
            ((inputReserves * outputAmount) * 10000) /
```

- Found in src/TSwapPool.sol Line: 402

```
            outputToken.safeTransfer(msg.sender, 1
                _000_000_000_000_000_000);
```

L[-6]: Unused Custom Error

it is recommended that the definition be removed when custom error is unused

1 Found Instances

- Found in src/PoolFactory.sol Line: 22

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

[L-7] `TSwapPool::LiquidityAdded` event has parameters out of order

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

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

Recommended Mitigation:

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

[L-8] 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, giving incorrect information to the caller.

Recommended Mitigation:

```
1      {
2          uint256 inputReserves = inputToken.balanceOf(address(this));
3          uint256 outputReserves = outputToken.balanceOf(address(this));
4
5  -         uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount
       , inputReserves, outputReserves);
6  +         output = getOutputAmountBasedOnInput(inputAmount,
       inputReserves, outputReserves);
7
8  -         if (output < minOutputAmount) {
9  -             revert TSwapPool__OutputTooLow(outputAmount,
       minOutputAmount);
10 +         if (output < minOutputAmount) {
11 +             revert TSwapPool__OutputTooLow(outputAmount,
       minOutputAmount);
12         }
13
14 -         _swap(inputToken, inputAmount, outputToken, outputAmount);
```

```
15 +         _swap(inputToken, inputAmount, outputToken, output);
16       }
```

Informational

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

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

[I-2] Lacking zero address checks

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

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

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

[I-4] 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.

- Found in src/TSwapPool.sol: Line: 44
- Found in src/PoolFactory.sol: Line: 37
- Found in src/TSwapPool.sol: Line: 46
- Found in src/TSwapPool.sol: Line: 43