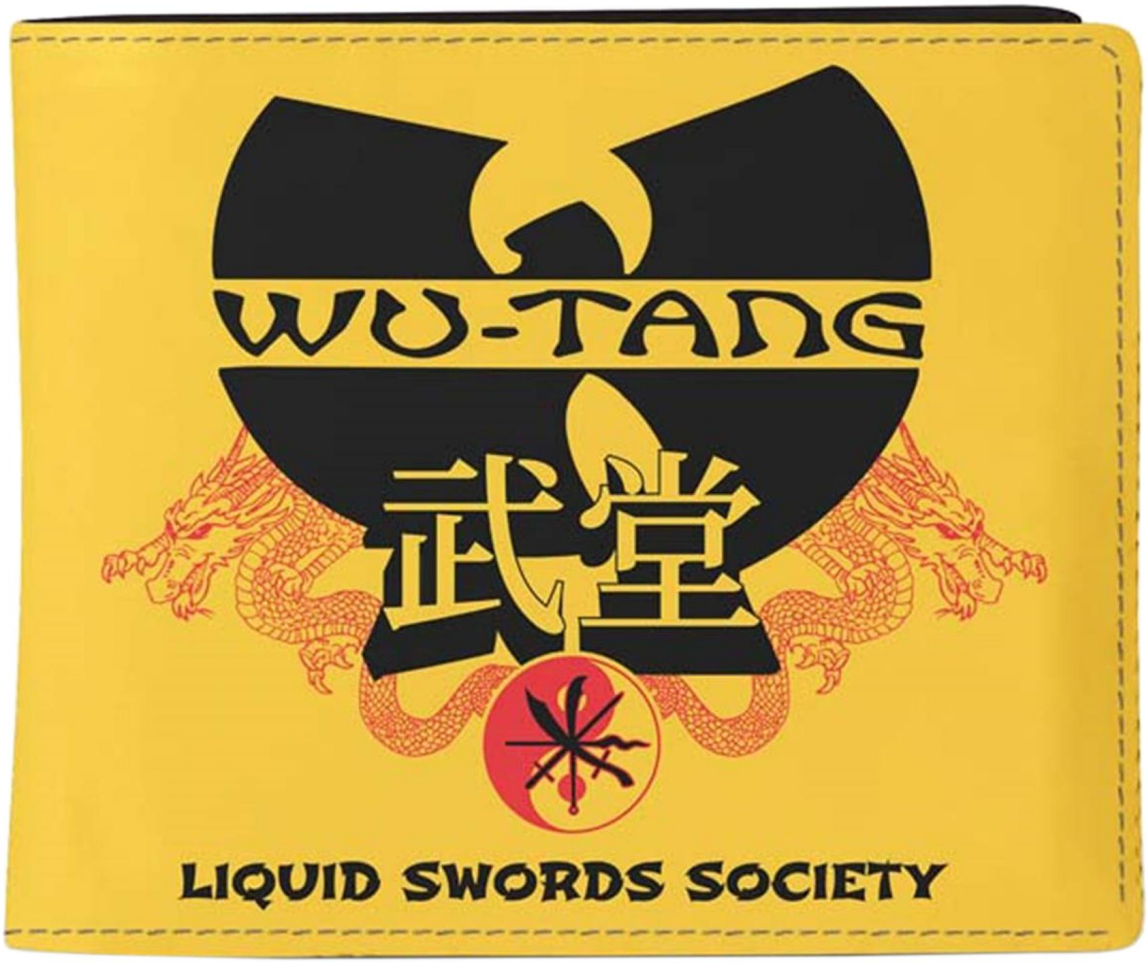

Protocol Audit Report

Prepared by: Georgi Penchev

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Georgi Penchev 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
	Low	M	M/L	L

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

Audit Details

Scope

- 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

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

Issues found

Severity	**Number of issues found**
High	4

Severity	Number of issues found
Medium	1
Low	3
Info	4
Gas	0
Total	12

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users. Resulting in lost fees.

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens the user should deposit given an amount of tokens of output tokens. However the function currently miscalculates the resulting amount. When calculating the 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) * 10000) /
+           ((inputReserves * outputAmount) * 1000) /
            ((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 it is done in `TSwapPoll::swapExactInput` where the function specifies `minOutputAmount`,

the `swapExactOutput` function should specify a `maxInputAmount`

Impact: If market condition change before transaction processes, the user should get a much wors 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: now
3. The function does not offer a max input amount
4. As the transaction is pending in the mempool the market changes! And the price moves HUGE -> 1 WETH is not 10 000 USDC, 10x more than the user expected.
5. The transction completes, but the user send the protocol 10 000 USDC then the expected 1000 USDC

Add this test to the `TSwapPool.t.sol`:

```
function testSwapExactOutput() public {
        uint256 initialLiquidity = 150e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 150e18);
        poolToken.approve(address(pool), 150e18);
        console.log("those amounts", initialLiquidity);
        pool.deposit({
            wethToDeposit: 150e18,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: 150e18,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

        address someUser = makeAddr("someUser");
        uint256 userInitialPoolTokenBalance = 40e18;
        poolToken.mint(someUser, userInitialPoolTokenBalance);
        vm.startPrank(someUser);
        uint256 amountBeforeSwap = poolToken.balanceOf(someUser);
        console.log("Before swap", amountBeforeSwap);

        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.timestamp));

        uint256 amountAfterSwap = poolToken.balanceOf(someUser);
        console.log("After swap", amountAfterSwap);
        uint256 amountDifference = amountBeforeSwap - amountAfterSwap;

        console.log("left", amountDifference);
        vm.stopPrank();

        poolToken.mint(address(pool), 1000 ether);
```

```
        vm.startPrank(someUser);
        uint256 amountBeforeSwap2 = poolToken.balanceOf(someUser);
        console.log("Before swap2", amountBeforeSwap2);

        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(
            poolToken,
            weth,
            0.2 ether,
            uint64(block.timestamp)
        );

        uint256 amountAfterSwap2 = poolToken.balanceOf(someUser);
        console.log("After swap2", amountAfterSwap2);
        vm.stopPrank();

        uint256 amountDifference2 = amountBeforeSwap2 - amountAfterSwap2;

        assertEq(amountDifference, amountDifference2);
    }
```

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,
        IERC20 outputToken,
        uint256 outputAmount,
+       uint256 maxInputAmount,
        uint64 deadline
.
.
.
    inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );

+       if(inputAmount > maxInputAmount){
+           revert();
+}

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

[H-3] `TSwapPool::sellPoolTokens` missmatches 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 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 should be called because users specify the exact amount of input tokens, not output.

Impact: The users will swap the wrong amount of tokens, which is severe disruption of protocol functionality.

Proof of Concept:

Recommended Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require the changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`)

```
function sellPoolTokens(
        uint256 poolTokenAmount,
+       uin256 minWethToReceive
    ) external returns (uint256 wethAmount) {
        return
-           swapExactOutput(
i_poolToken,i_wethToken,poolTokenAmount,uint64(block.timestamp));
+
swapExactInput(i_poolToken,poolTokenAmount,i_wethToken,minWethToReceive,uint64(blo
ck.timestamp))
    }
```

[H-4] In `TSwapPool::_swap` the exgra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`

Description: The protocol follows the strict invariant `x * y =k` Where:

- `x`: The balance of of the pool token
- `y`: The balance of weth
- `k`: The constant product of the two balances

This means the whenever the balance change in the protocol the ration between the 2 amount should be constant, hence the `k`. However this is broken due to the extra incentive in the `_swap` function. Meaning that slowly over time the protocol funds will be drained.

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

Impact: The user could maliciously drain the protocol of funds by doing a lot of swaps and collecting extra incentive given out by the protocol.

Proof of Concept:

1. A user swaps 10 times and collect the extra incentive of `1_000_000_000_000_000_000` tokens
2. The user continues to swap untill all the protocol funds are drained

▶ Proof Of Code

```
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;

        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

        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)
        );
        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 or should account the incentive in the calculation.

Low

[L-1] TITLE `TSwapPool::LiquidityAdded` has parameters out of order causing event to emit incorrect information

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

Impact: Event emission in incorrent, leading to off-chain functions poterntially malfunctioning.

Recommended Mitigation:

```
- 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 output it is never assigned a value

Impact: The return value will always be zero, given incorrect information to the caller.

Recommended Mitigation:

```
    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);
```

Medium

[M-1] `TSwapPool::deposit()` missing deadline check causing transactions to complete even after deadline

Description: The `deposit()` function accepts a deadline parameter, which according to the documentation is "@param deadline The deadline for the transaction to be completed by" . However this parameter is not used. As concequence, operations that add liquidity to the pool might be executed at unexpected times, in market conditions when the deposit rate is not favoritable

Impact: Transaction can be sent when market conditions are not favoritable to deposit, even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is not used.

Recommended Mitigation: Consider making the following changes to the function

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, //Liquidity pool tokens -> if empty,
we can pick 100% (100% == 17 tokents) example
        uint256 maximumPoolTokensToDeposit,
```

```
        uint64 deadline
    )
        external
        revertIfZero(wethToDeposit)
+       revertIfDeadlinePassed(deadline)
        returns (uint256 liquidityTokensToMint)
    {
```

Informational

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

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

[I-2] Lacking zero address checks

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

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

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

▶ 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(
```