title: TSwap Protocol Audit Report author: skid0016 date: June 17, 2024-=header-includes:

- \usepackage
- \usepackage

Prepared by: [skid0016] Lead Auditors:

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

skid0016 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 (https://docs.codehawks.com/hawks-auditors/how-to-evaluate-a-finding-severity) severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

Roles

Executive Summary

Issues found

Severtity	Number of issues found
High	4
Medium	2
Low	2
Info	9
Total	17

Findings

[H-1] Incorrect fee calculation in `TSwapPoll::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost funds

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 the users

Proof of Concept:(WRITE A POC)

Recommended Mitigation:

```
        function getInputAmountBasedOnOutput(
         uint256 outputAmount,
         uint256 inputReserves,
         uint256 outputReserves
     )
         public
         pure
         revertIfZero(outputAmount)
         revertIfZero(outputReserves)
         returns (uint256 inputAmount)
     {

-         return
-             ((inputReserves * outputAmount) * 10000) /
-             ((outputReserves - outputAmount) * 997);
+         return
+             ((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 is done `TSwapPoll::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:

I tried to create a scenerio where someone swap without slippage protection.

I eddited the function and added a `maxInputAmount` and wrote an error to revert if the `inputAmount > maxInputAmount`:

```
     function swapExactOutput(
    IERC20 inputToken,
    IERC20 outputToken,
    uint256 outputAmount,
    uint256 maxInputAmount,
    uint64 deadline
)
    public
    revertIfZero(outputAmount)
    revertIfZero(maxInputAmount)
    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 TSwapPool__InputAmountTooHigh(inputAmount, maxInputAmount);
    }

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

Add this test to your TSwapPool.t.sol

```
    function testSwapExactOutputWithoutSlippageProtection() public {
        // Liquidity provider deposits tokens into the pool
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        // User attempts to swap tokens with high slippage
        vm.startPrank(user);
        uint256 outputWeth = 1e18;  // Output WETH desired
        uint256 poolTokenAmount = pool.getInputAmountBasedOnOutput(outputWeth, poolToken.balanceOf(address(pool)), weth.balanceOf(add

        // Approve the pool to spend user's pool tokens
        poolToken.approve(address(pool), poolTokenAmount);

        // Attempt the swap
        console2.log("Transaction should revert due to slippage.");
        try pool.swapExactOutput(poolToken, weth, outputWeth, poolTokenAmount, uint64(block.timestamp)) {
            revert("Transaction did not revert due to slippage.");
        } catch {
            console2.log("Transaction reverted as expected due to slippage.");
        }
        vm.stopPrank();
    }
```

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

+    if (inputAmount > maxInputAmount) {
+        revert TSwapPool__InputAmountTooHigh(inputAmount, +maxInputAmount);
    }
    );
```

[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 miscalutes 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 token , which is a 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 changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`)

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

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

Description: The protocol follows a strick 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 whatever the balances change in the protocol, the ratio between the two amounts should remain constant, hence `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 following 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

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++;
-        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 dealine check causing transaction to

complete even after the dealine

Description: The `deposit` function accepts a dealine 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, operations that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact: Transaction could be sent when market conditions are unfavorable to deposit, even when adding a dealine paprameter.

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

Recommended Mitigation: COnsider making the following change in the function

```
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, // LP tokens => if empty , we can pick 100%(100% == 17 tokens)
        uint256 maximumPoolTokensToDeposit,
        // @audit-HIGH , deadline not being used
        // if someone sets a deadline, let's say, next block
        // they could still deposit
        // IMPACT: HIGH  a user expects a deposit to fail, will go through . Severe disruption of functionality
        // Likelihood: HIGH
        uint64 deadline
    )
        external
+       revertIfDealinePassed(dealine)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
```

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

Description: “The function increments the `swap_count` variable with each transaction and checks if `swap_count` has reached a specified maximum (`SWAP_COUNT_MAX`). Once this threshold is met, the `swap_count` is reset to 0, and `outputToken` is transferred to the sender. However, the use of fee-on-transfer tokens can disrupt this logic, breaking the protocol's invariant and leading to unexpected behaviors.”

Impact: “When fee-on-transfer tokens are involved, the amount transferred to the sender may not match the intended value. This discrepancy can result in the protocol miscalculating the `swap_count` and performing transfers that are inconsistent with the expected behavior. After 10 transactions, the system might fail to correctly handle the fee-on-transfer mechanism, leading to inaccurate token transfers and potential loss of funds or incorrect state changes.”

Proof of Concept:

```
    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); // Transfer 1 token
}
```

1. Perform a swap operation 10 times.
2. Each swap increments `swap_count`.
3. Upon reaching the 10th swap, `swap_count` is reset to 0, and `outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000)` is called.
4. If `outputToken` is a fee-on-transfer token, the actual amount received by `msg.sender` may be less than the intended 1 token due to the transfer fee.”

Recommended Mitigation:

1. Avoid Direct Transfers with Fee-on-Transfer Tokens: Implement logic to account for fee-on-transfer tokens. Use a pre-transfer and post-transfer balance check to determine the actual amount received by the recipient.
2.

Use Standard Tokens or Adjust Logic: Ensure the protocol either avoids fee-on-transfer tokens or adjusts the transfer logic to handle such tokens correctly. 3. Audit and Test with Various Token Types: Perform extensive testing with different types of tokens (including fee-on-transfer and ERC777 tokens) to ensure compatibility and correct handling of all edge cases.

Low

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

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

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

[L-2] Default value returned by `TSwapPool:swapExactInput` results in incorrect 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, nor uses an explicit return statement.

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

Proof of Concept:

Recommended Mitigation:

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

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

+        uint256 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);
    }
```

Recommended Mitigation:

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

Informational

[I-1] `PoolFactory::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;
    }
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+      if(wethToken = address(0)){
+        revert();
+      }

+      if(poolToken = address(0)){
+        revert();
+      }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

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

```
-  string memory liquidityTokenName = string.concat(
-           "T-Swap ",
-           IERC20(tokenAddress).name()
-   );

+   string memory liquidityTokenName = string.concat(
+           "T-Swap ",
+           IERC20(tokenAddress).name()
+    );
```

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/PoolFactory.sol Line: 35 (src/PoolFactory.sol#L35)

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

- Found in src/TSwapPool.sol Line: 52 (src/TSwapPool.sol#L52)

```
      event LiquidityAdded()
```

- Found in src/TSwapPool.sol Line: 57 (src/TSwapPool.sol#L57)

```
      event LiquidityRemoved()
```

- Found in src/TSwapPool.sol Line: 62 (src/TSwapPool.sol#L62)

```
      event Swap()
```