Protocol Audit Report

Version 1.0

Jaunepr

2024 年 7 月 30 日

Protocol Audit Report

Jaunepr

May 24, 2024

Prepared by: Jaunepr Lead Auditors: - Jaunepr

Table of Contents

    * [H-4] `TSwapPool::_swap`函数中，每 10 次交易会给调用者额外的 token 奖励，这打破了协议的 Invariant：`x * y = k`
    * [H-5] `TSwapPool::swapExactOutput`函数缺少边界保护，可能导致用户获得更少的 Token。
  - Low
    * [L-1] `TSwapPool::_addLiquidityMintAndTransfer`函数中的触发`LiquidityAdded`事件中的参数顺序错误
    * [L-2] `TSwapPool::swapExactInput` 的返回值没有使用
  - Informationals
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` 未使用过，应该移除
    * [I-2] `PoolFactory`缺少零地址检查
    * [I-3] `PoolFavtory::liquidityTokenSymbol` 该变量应该调用 `.symbol()` 而不是 `.name()`
    * [I-4]: **`public`** functions not used internally could be marked `external`
    * [I-5]: Define and use `constant` variables instead of using literals
    * [I-6]: Event is missing `indexed` fields
    * [I-7]: PUSH0 is not supported by all chains
    * [I-8]: Large literal values multiples of 10000 can be replaced with scientific notation
    * [I-9]: Unused Custom Error # Protocol Summary

Protocol description blablabla…

Disclaimer

(Blablabla) The YOUR_NAME_HERE 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

		Impact		
	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

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

```
Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

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

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

审查过程摘要

Issues found

严重程度	问题个数
高	5
中	0
低	2
提示	9
共计	16

Findings

High

[H-1] `TSwapPool::deposit` deadline varliable never being used, that cause deposit can be used after the deadline.

Description:`TSwapPool::deposit`函数接收了一个`deadline`参数,根据注释，它意味着交易的截止时间。但在整个函数中却未被使用，因此在存款利率不利的市场条件下，向资金池添加流动性的操作可能会在意想不到的时间被执行。

Impact: 当超过截止时间，市场条件不利时，仍然可以发送交易。

Recommended Mitigation:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[H-2] 在 `TSwapPool::getInputAmountBasedOnOutput` 函数计算结果时，费率错误，会导致用户大量资金损失

Description: 在`getInputAmountBasedOnOutput`函数是通过给定用户需要取出的 Token 数量计算用户需要存入的 Token 数量。但是在该函数中，计算的 fee 率有误，应该是 `1-997/1000 = 1 -`

`99.7% = 0.3%，而不是1 - 997/10000 = 1 - 9.97% = 90.03%。`

Impact: 合约收取了用户大量费用。

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-3] TSwapPool::sellPoolTokens 中弄混了该调用的函数应该是 swapExactInput 而非 swapExactOutput. 会导致用户收到错误数量的 outputToken。

Description: `sellPoolTokens`根据注释得知，该函数给定数量的 inputToken，来交换实际获得相对数量的 Weth. 所以在其中应该调用的函数是`swapExactInput`而不是`swapExactOutput`。

Impact: 用户会获得错误数量的 outputToken，破坏合约的功能。

Recommended Mitigation:

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

[H-4] TSwapPool::_swap 函数中，每 10 次交易会给调用者额外的 token 奖励，这打破了协议的 Invariant：`x * y = k`

Description: 协议遵守`x * y = k`Invariant，其中 - `x`: 池中 poolToken 余额 - `y`: 池中的 Weth 余额 - `k`: 两个余额的常数乘积无论池子里的两种币的余额如何变化，始终保持比率`k`不变。而`_swap`函数的额

外奖励机制会打破这种不变性，导致最终协议的资金耗尽。以下是_swap函数中造成这种问题的代码：

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

Impact: 用户可以刻意的操纵交易数量来耗尽协议里的资金。并且协议的核心不变量破坏。

Proof of concept: 1. 当一个用户交易了 10 次，得到了1_000_000_000_000_000_000token 的额外奖励; 2. 用户继续交易，直到协议资金耗尽；

Proof of Code

将下述代码添加至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: 1. 修改公式保持不变量不被破坏。2. 移除奖励机制。

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

[H-5] `TSwapPool::swapExactOutput` 函数缺少边界保护，可能导致用户获得更少的 Token。

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);
```

Low

[L-1] `TSwapPool::_addLiquidityMintAndTransfer` 函数中的触发 `LiquidityAdded` 事件中的参数顺序错误

Description: 当`LiquidityAdded`事件触发后，日志会错误排序，`poolTokensToDeposit`和`wethToDeposit`的位置互换了。

Impact: 当日志中的数据不对，可能会导致链下业务跟着出错

Recommended Mitigation:

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

[L-2] `TSwapPool::swapExactInput` 的返回值没有使用

Description: `swapExactInput`声明了返回值 output，意思是返回用户购买的 token 数量，但 output 变量在函数中并未使用。根据对代码的解读，`output`应该改为`outputAmout`。

Impact: 返回值会永远等于 0，会导致调用者误以为返回值是 0。

Recommended Mitigation:

```
    function swapExactInput(
            IERC20 inputToken,
            uint256 inputAmount,
            IERC20 outputToken,
            uint256 minOutputAmount,
            uint64 deadline
        )
            public
            revertIfZero(inputAmount)
            revertIfDeadlinePassed(deadline)
            returns (
-               uint256 output
+               uint256 outputAmount
            )
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

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

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

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

Informationals

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` 未使用过，应该移除

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

[I-2] `PoolFactory` 缺少零地址检查

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

[I-3] `PoolFavtory::liquidityTokenSymbol` 该变量应该调用 `.symbol()` 而不是 `.name()`

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

[I-4]: `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(
  ```

[I-5]: 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: 457

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 466

```
                1e18,
```

[I-6]: 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(
```

[I-7]: 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;
  ```

[I-8]: 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: 405

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

[I-9]: 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

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