tags: `Preliminary Report`

Fountain LPOracleAnchoredView Audit – Preliminary Issue Analysis

Feb 24, 2022
by **Verilog Audit**

This is the preliminary audit report to summarize the **most urgent and critical issues** found by the Verilog Audit. We will list the major concerns and findings here. Once the issues below are all resolved, we can proceed to a lower-level security and improvement analysis.

Table of Content

Summary of the Incremental Audit

We audited Fountain Protocol and the previous auditing report is here (https://hackmd.io/C_lPwlT0TsuONfUWkQzfpw) with hash cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/tree/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

This is the incremental audit for file `LPOracleAnchoredView.sol` (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol). The main functionality added is the calculation for LP token price.

Privileged Roles

1. The caller of the contractor has the privileged role to determine the token pairs and their token symbol and address.

Findings & Improvement Suggestions

Informational **Minor** **Medium** **Major** **Critical**

	Total	Acknowledged	Resolved
Critical	0	0	0
Major	0	0	0
Medium	0	0	0
Minor	2	2	0
Informational	3	3	0

Critical

none ;)

Major

none ;)

Medium

none ;)

Minor

1. The values in the array of input to the constructor (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol#L49) of contract `LPOracleAnchoredView` are not checked. **minor**
   Description: Two potential issues are here. First, the function does not prevent the situation that the input may contain the same token symbols. Second, the address of the token may not be a legitimate token address.

```
constructor(address _ref,OracleTokenConfig[] memory configs) public {
    ref = IStdReference(_ref);
    for(uint i = 0; i < configs.length; i++){
        OracleTokenConfig memory config = configs[i];
        require(config.baseUnit > 0, "baseUnit must be greater than zero");
        CTokenConfigs[config.symbol] = config;
        cTokenSymbol[config.cToken] = config.symbol;
    }
}
```

Recommendation: We suggest two changes should be made in the constructor. First, before assigning the configuration to `CTokenConfigs[config.symbol]`, the `CTokenConfigs[config.symbol]` should not contain any value. Second, we suggest adding the requirement `require (IERC20(config.cToken).symbol()==config.symbol)` in the constructor to double-check if the token address is an ERC20 token and make sure the symbol is exactly the same.

Result: Pending...

2. The token decimal alignment is not needed in function `reserveProductAndTotalSupply()` (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol#L91). Furthermore, doing such an alignment may cause a loss of accuracy if one of the tokens in the pair has a decimal value greater than 18. **minor**
Description: In the AMM model, to determine the product of the amounts of two tokens in a pair, we do not need to cast the decimal to 18. Besides, doing this kind of casting may result in the loss of accuracy if one of the tokens in a pair has the decimal greater than 18.

```
function reserveProductAndTotalSupply(string memory symbol) internal view re
    OracleTokenConfig memory config = CTokenConfigs[symbol];
    IDexPair dexPair = IDexPair(config.underlying);
    totalSUpply = dexPair.totalSupply();
    (uint112 reserve0, uint112 reserve1,) = dexPair.getReserves();
    uint decimal0 = OracleERC20(dexPair.token0()).decimals();
    uint decimal1 = OracleERC20(dexPair.token1()).decimals();
    uint amount0 = uint(reserve0).mul(1e18).div(10 ** decimal0);
    uint amount1 = uint(reserve1).mul(1e18).div(10 ** decimal1);
    product = amount0.mul(amount1);
}
```

Recommendation: We suggest to remove the decimal alignment or do not use this contract for token with decimals greater than 18 otherwise there can be loss of accuracy

Result: Pending...

Informational

1. Typo in function return values **Informational**
   Description: There is a typo in the return values of function `reserveProductAndTotalSupply()` (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol#L91). The `totalSUpply` should be spelled like `totalSupply`. The variables should be spelled in *camel* format.
   Recommendation: `totalSUpply -> totalSupply`.
   Result: Pending...

2. Magic Numbers
   Description: There are some *magic numbers* in the code deck. For example, `1e28` in Line 67 (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol#L67), `1e10` in Line 82 (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol#L82) and Line 87 (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol#L87).
   Informational
   Recommendation: Make these magic numbers constant values with comments.
   Result: Pending...

3. Unnecessary ordering between `tokenA` and `tokenB` in function `_priceProduct()` (https://github.com/dev-fountain/fountain-protocol/blob/e2c39e77c4df4a93f807abdd546a40ff76a7b483/contracts/LPOracleAnchoredView.sol#L103).
   Informational
   Description: In this function, it purposely changes the order of `tokenA` and `tokenB` and saves the symbol of the token that has a smaller token address in variable `symbol0` and the other in `symbol1`. does not differentiate the orders. The orders of the tokens does not differentiate the result of product (`product = price0.mul(price1)`).

```
function priceProduct(string memory symbol) internal view returns(uint produ
    OracleTokenConfig memory config = CTokenConfigs[symbol];
    string memory symbol0;
    string memory symbol1;
    if(config.tokenA < config.tokenB){
        symbol0 = config.symbolA;
        symbol1 = config.symbolB;
    }else{
        symbol0 = config.symbolB;
        symbol1 = config.symbolA;
    }
    uint price0 = oraclePrice(symbol0).rate;
    uint price1 = oraclePrice(symbol1).rate;
    product = price0.mul(price1);
}
```

Recommendation: Token symbols can be assigned directly without checking the token orders.
Result: Pending...