MythX

Analysis 13aaa49c-fcee-4436-a2ef-bda037c55ae6 MythX

Started	
Finished	Mon Jul 03 2023 19:04:06 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Mythx-Cli-0.7.3
Main Source File	Contracts/SilicaEthStaking.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	2

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.8.6"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/SilicaEthStaking.sol

Locations

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.6;

import {AbstractSilicaV2_1} from "./AbstractSilicaV2_1.sol";
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/SilicaEthStaking.sol`

Locations

```
function getLastIndexedDay() internal view override returns (uint32) {
IOracleEthStaking oracleEthStaking = IOracleEthStaking(
IOracleRegistry(oracleRegistry).getOracleAddress(address(rewardToken), COMMODITY_TYPE)
);
uint32 lastIndexedDayMem = oracleEthStaking.getLastIndexedDay();
```

Source file

`contracts/SilicaEthStaking.sol`

Locations

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract SilicaEthStaking is AbstractSilicaV2_1 {
uint8 public constant COMMODITY_TYPE = 2;

function decimals() public pure override returns (uint8) {
return 18;
}

constructor() ERC20("Silica", "SLC") {}

/// @notice Function to return the last day silica is synced with Oracle
function getLastIndexedDay() internal view override returns (uint32) {
IOracleEthStaking oracleEthStaking = IOracleEthStaking(
IOracleRegistry(oracleRegistry).getOracleAddress(address(rewardToken), COMMODITY_TYPE)
);
uint32 lastIndexedDayMem = oracleEthStaking.getLastIndexedDay();
require(lastIndexedDayMem != 0, "Invalid State");

return lastIndexedDayMem;
}

/// @notice Function to return the amount of rewards due by the seller to the contract on day inputed
function getRewardDueOnDay(uint256 _day) internal view override returns (uint256) {
IOracleEthStaking oracleEthStaking = IOracleEthStaking(
IOracleRegistry(oracleRegistry).getOracleAddress(address(rewardToken), COMMODITY_TYPE)
);
(, uint256 baseRewardPerIncrementPerDay, , , , , ) = oracleEthStaking.get(_day);

return RewardMath.getEthStakingRewardDue(totalSupply(), baseRewardPerIncrementPerDay, decimals());
}

/// @notice Function to return an array with the amount of rewards due by the seller to the contract on days in range inputed
function getRewardDueInRange(uint256 _firstDay, uint256 _lastDay) internal view override returns (uint256[] memory) {
IOracleEthStaking oracleEthStaking = IOracleEthStaking(
IOracleRegistry(oracleRegistry).getOracleAddress(address(rewardToken), COMMODITY_TYPE)
);
uint256[] memory baseRewardPerIncrementPerDayArray = oracleEthStaking.getInRange(_firstDay, _lastDay);

uint256[] memory rewardDueArray = new uint256[](baseRewardPerIncrementPerDayArray.length);

uint8 decimalsMem = decimals();
uint256 totalSupplyCopy = totalSupply();
for (uint256 i = 0; i < baseRewardPerIncrementPerDayArray.length; i++) {
```

```
rewardDueArray[i] = RewardMath.getEthStakingRewardDue(totalSupplyCopy, baseRewardPerIncrementPerDayArray[i], decimalsMem);
}

return rewardDueArray;
}

/// @notice Returns the commodity type the seller is selling with this contract
/// @return The commodity type the seller is selling with this contract
function getCommodityType() external pure override returns (uint8) {
return COMMODITY_TYPE;
}

/// @notice Returns decimals of the contract
function getDecimals() external pure override returns (uint8) {
return decimals();
}
}
```