Moonwell Finance - Safety Module

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 8th, 2021 - February 13th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/11/2022	Ataberk Yavuzer
0.2	Document Edits	02/12/2022	Ataberk Yavuzer
0.3	Draft Review	02/13/2022	Gabi Urrutia
1.0	Remediation Plan	02/21/2022	Ataberk Yavuzer
1.1	Remediation Plan Review	02/21/2022	Gabi Urrutia
1.2	Remediation Plan Edit	02/23/2022	Ataberk Yavuzer
1.2	Remediation Plan Edit Review	02/23/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Moonwell Finance engaged Halborn to conduct a security audit on their smart contracts beginning on February 8th, 2021 and ending on February 13th, 2021 . The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided one week for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were addressed by the Moonwell Finance team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Dynamic Analysis (ganache-cli, brownie, hardhat)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Moonwell Finance Safety Module Contracts

    (a) Repository: Safety Module

    (b) Commit ID: 827bbd043ef7b876f06d81044ce17052d4bd820e

2. Out-of-Scope

    (a) test/*.sol

    (b) utils/*.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	3

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨	🟥	🟥
🟩	🟩	🟨	🟨	🟥
⬜	🟩 (HAL-01) (HAL-02)	🟩	🟨	🟨
⬜ (HAL-03) (HAL-04) (HAL-05)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING ZERO ADDRESS CHECKS	Low	SOLVED - 02/18/2022
(HAL-02) MISSING REENTRANCY GUARD	Low	SOLVED - 02/18/2022
(HAL-03) USE 1E18 CONSTANT FOR GAS OPTIMIZATION	Informational	SOLVED - 02/18/2022
(HAL-04) USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION	Informational	SOLVED - 02/18/2022
(HAL-05) USE IMMUTABLE KEYWORD FOR GAS OPTIMIZATION	Informational	NOT APPLICABLE

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING ZERO ADDRESS CHECKS - LOW

Description:

Safety Module contracts have address fields on multiple functions. These functions have missing address validations. Every address should be validated and checked that is different from zero. This is also considered a best practice.

During the test, it has seen some of these inputs are not protected against using the address(0) as the target address.

Code Location:

Listing 1: StakedToken.sol (Lines 50,58)

```
44 function __StakedToken_init(
45         IERC20 stakedToken,
46         IERC20 rewardToken,
47         uint256 cooldownSeconds,
48         uint256 unstakeWindow,
49         address rewardsVault,
50         address emissionManager,
51         uint128 distributionDuration,
52         string memory name,
53         string memory symbol,
54         uint8 decimals,
55         address governance
56     ) internal initializer {
57         __ERC20_init_unchained(name, symbol, decimals);
58         __DistributionManager_init_unchained(emissionManager,
              distributionDuration);
59         __StakedToken_init_unchained(
60             stakedToken,
61             rewardToken,
62             cooldownSeconds,
63             unstakeWindow,
64             rewardsVault,
65             governance
66          );
67      }
```

Listing 2: EcosystemReserve.sol (Line 32)

```
31 function initialize(address reserveController) external
       initializer {
32          _setFundsAdmin(reserveController);
33      }
```

Listing 3: EcosystemReserve.sol (Line 52)

```
51 function setFundsAdmin(address admin) external override
       onlyFundsAdmin {
52          _setFundsAdmin(admin);
53      }
```

Recommendation:

It is recommended to validate that every address input is different from zero.

Remediation Plan:

SOLVED: Moonwell Finance team solved this issue by adding multiple zero address checks to the code.

Commit ID: **4010c55a1103aa7ebdfe47a2a2f4ac697507ba8f**

3.2 (HAL-02) MISSING REENTRANCY GUARD - LOW

Description:

To protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against re-entrancy attacks.

Code Location:

Listing 4: EcosystemReserve.sol

```
function transfer(
        IERC20 token,
        address recipient,
        uint256 amount
    ) external override onlyFundsAdmin {
        token.transfer(recipient, amount);
    }
```

Listing 5: StakedToken.sol

```
function stake(address onBehalfOf, uint256 amount) external override {
        require(amount != 0, 'INVALID_ZERO_AMOUNT');
        require(onBehalfOf != address(0), 'STAKE_ZERO_ADDRESS');
        uint256 balanceOfUser = balanceOf(onBehalfOf);
```

Listing 6: StakedToken.sol

```
function redeem(address to, uint256 amount) external override {
        require(amount != 0, 'INVALID_ZERO_AMOUNT');
        require(to != address(0), 'REDEEM_ZERO_ADDRESS');
        //solium-disable-next-line
        uint256 cooldownStartTimestamp = stakersCooldowns[msg.
            sender];
```

Listing 7: StakedToken.sol

```
function claimRewards(address to, uint256 amount) external
    override {
        uint256 newTotalRewards = _updateCurrentUnclaimedRewards(
            msg.sender,
            balanceOf(msg.sender),
            false
        );
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

The functions on the code location section have missing nonReentrant modifiers. It is recommended to add OpenZeppelin ReentrancyGuard library to the project and use the nonReentrant modifier to avoid introducing future re-entrancy vulnerabilities.

Remediation Plan:

SOLVED: This issue was solved by implementing nonReentrant modifier to specified functions above.

Commit ID: **ff3f500f9ea1a340cc1b434ff306be9da94925f1**

3.3 (HAL-03) USE 1E18 CONSTANT FOR GAS OPTIMIZATION - INFORMATIONAL

Description:

In Solidiy, exponentiation operation (**) costs up to 10 gas. It is possible to consume less gas to calculate token prices if DECIMAL variable is fixed.

Code Location:

Listing 8: DistributionManager.sol

```
return principalUserBalance.mul(reserveIndex.sub(userIndex)).div
    (10**uint256(PRECISION));
```

Listing 9: DistributionManager.sol

```
emissionPerSecond.mul(timeDelta).mul(10**uint256(PRECISION)).div(
    totalBalance).add(currentIndex);
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use 1e18 instead of (10 ** 18) on price calculations to optimize gas usage.

Remediation Plan:

SOLVED: Moonwell Finance team solved this issue by using 1e18 instead of (10 ** PRECISION) on the code.

Commit ID: **55a3c2ca180a7463cfda215ca594a02d7c08fa6f**

Example PoC:

Listing 10: Gas Usage Comparison

```
pragma solidity 0.8.7;
contract Null{
    constructor() {}
    function test() public view returns (uint256) {
        return 10 ** 18; // gas usage: 21407
    }
    function test2() public view returns (uint256) {
        return 1e18; // gas usage: 21379
    }
}
```

3.4 (HAL-04) USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION - INFORMATIONAL

Description:

In all the loops, the variable i is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++. This also affects variables incremented inside the loop code block.

Code Location:

Listing 11: DistributionManager.sol

```
for (uint256 i = 0; i < assetsConfigInput.length; i++) {
            AssetData storage assetConfig = assets[
                assetsConfigInput[i].underlyingAsset];
```

Listing 12: DistributionManager.sol

```
for (uint256 i = 0; i < stakes.length; i++) {
      accruedRewards = accruedRewards.add(
        _updateUserAssetInternal(
          user,
          stakes[i].underlyingAsset,
          stakes[i].stakedByUser,
          stakes[i].totalStaked
        )
      );
```

Listing 13: DistributionManager.sol

```
for (uint256 i = 0; i < stakes.length; i++) {
      AssetData storage assetConfig = assets[stakes[i].
          underlyingAsset];
      uint256 assetIndex = _getAssetIndex(
        assetConfig.index,
        assetConfig.emissionPerSecond,
        assetConfig.lastUpdateTimestamp,
        stakes[i].totalStaked
      );
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of an uint variable inside a loop. This also applies to the variables declared inside the for loop, not just the iterator. On the other hand, this is not applicable outside of loops.

Remediation Plan:

SOLVED: Moonwell Finance team solved this issue by replacing post-increment to pre-increment.

Commit ID: **4c95dae92775eedb8bb7f9e8ad876fa9518229ac**

3.5 (HAL-05) USE IMMUTABLE KEYWORD FOR GAS OPTIMIZATION - INFORMATIONAL

Description:

The immutable keyword was added to Solidity in 0.6.5. State variables can be marked immutable which causes them to be read-only, but only assignable in the constructor. The following state variables are missing the immutable modifier:

- STAKED_TOKEN
- REWARD_TOKEN
- COOLDOWN_SECONDS

There are no setter functions to change values of variables above. If these variables do not need to be changed, they should be defined with immutable keyword.

Code Location:

Listing 14: StakedToken.sol

```
IERC20 public STAKED_TOKEN;
IERC20 public REWARD_TOKEN;
uint256 public COOLDOWN_SECONDS;

/// @notice Seconds available to redeem once the cooldown period is fulfilled
uint256 public UNSTAKE_WINDOW;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add the immutable modifier to the state variables mentioned to save some gas.

Remediation Plan:

NOT APPLICABLE: Immutable variables can only be initialized inline or assigned directly in the constructor. In this finding, there is no explicit constructor in initializable smart contracts. Attempting to set the immutable keyword for variables specified above may break their operation.

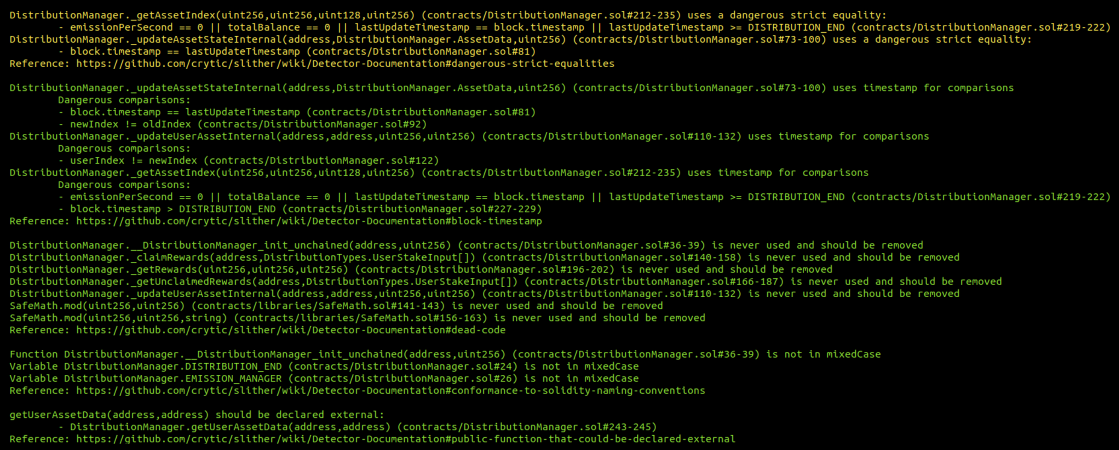
AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

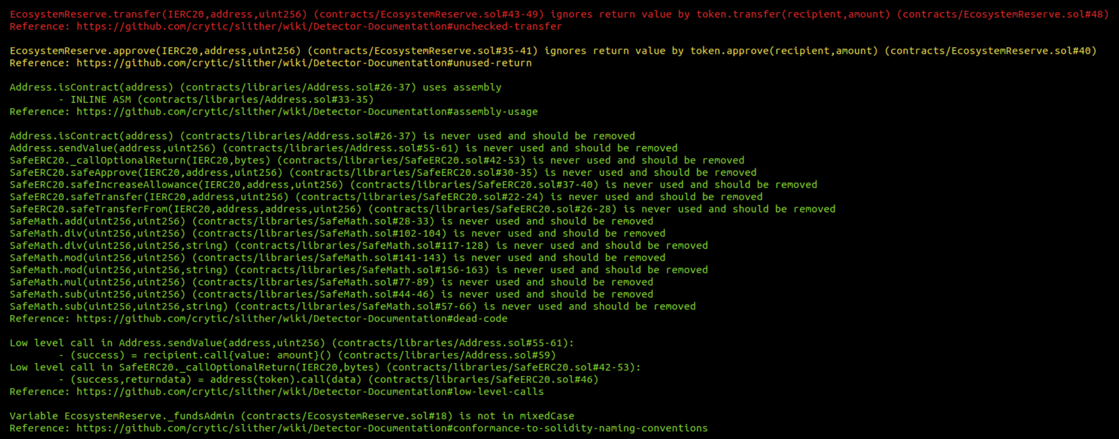
Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

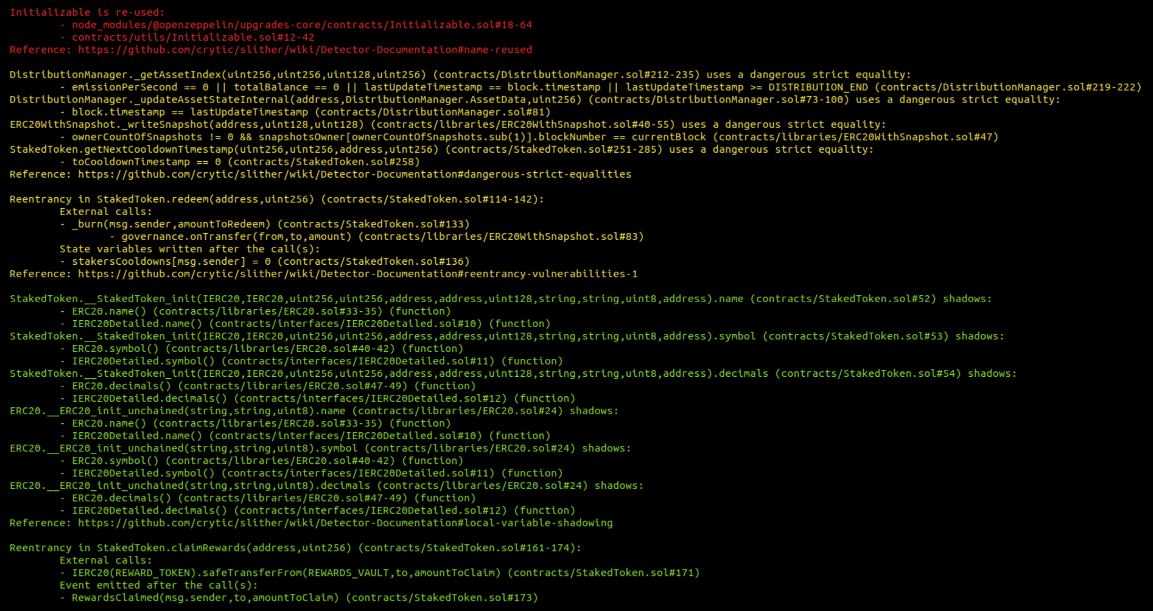
Results:



```
DistributionManager._getAssetIndex(uint256,uint256,uint128,uint256) (contracts/DistributionManager.sol#212-235) uses a dangerous strict equality:
        - emissionPerSecond == 0 || totalBalance == 0 || lastUpdateTimestamp == block.timestamp || lastUpdateTimestamp >= DISTRIBUTION_END (contracts/DistributionManager.sol#219-222)
DistributionManager._updateAssetStateInternal(address,DistributionManager.AssetData,uint256) (contracts/DistributionManager.sol#73-100) uses a dangerous strict equality:
        - block.timestamp == lastUpdateTimestamp (contracts/DistributionManager.sol#81)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

DistributionManager._updateAssetStateInternal(address,DistributionManager.AssetData,uint256) (contracts/DistributionManager.sol#73-100) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp == lastUpdateTimestamp (contracts/DistributionManager.sol#81)
        - newIndex != oldIndex (contracts/DistributionManager.sol#92)
DistributionManager._updateUserAssetInternal(address,address,uint256,uint256) (contracts/DistributionManager.sol#110-132) uses timestamp for comparisons
        Dangerous comparisons:
        - userIndex != newIndex (contracts/DistributionManager.sol#122)
DistributionManager._getAssetIndex(uint256,uint256,uint128,uint256) (contracts/DistributionManager.sol#212-235) uses timestamp for comparisons
        Dangerous comparisons:
        - emissionPerSecond == 0 || totalBalance == 0 || lastUpdateTimestamp == block.timestamp || lastUpdateTimestamp >= DISTRIBUTION_END (contracts/DistributionManager.sol#219-222)
        - block.timestamp > DISTRIBUTION_END (contracts/DistributionManager.sol#227-229)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

DistributionManager.__DistributionManager_init_unchained(address,uint256) (contracts/DistributionManager.sol#36-39) is never used and should be removed
DistributionManager._claimRewards(address,DistributionTypes.UserStakeInput[]) (contracts/DistributionManager.sol#140-158) is never used and should be removed
DistributionManager._getRewards(uint256,uint256,uint256) (contracts/DistributionManager.sol#196-202) is never used and should be removed
DistributionManager._getUnclaimedRewards(address,DistributionTypes.UserStakeInput[]) (contracts/DistributionManager.sol#166-187) is never used and should be removed
DistributionManager._updateUserAssetInternal(address,address,uint256,uint256) (contracts/DistributionManager.sol#110-132) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/libraries/SafeMath.sol#141-143) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/libraries/SafeMath.sol#156-163) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Function DistributionManager.__DistributionManager_init_unchained(address,uint256) (contracts/DistributionManager.sol#36-39) is not in mixedCase
Variable DistributionManager.DISTRIBUTION_END (contracts/DistributionManager.sol#24) is not in mixedCase
Variable DistributionManager.EMISSION_MANAGER (contracts/DistributionManager.sol#26) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

getUserAssetData(address,address) should be declared external:
        - DistributionManager.getUserAssetData(address,address) (contracts/DistributionManager.sol#243-245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```



```
EcosystemReserve.transfer(IERC20,address,uint256) (contracts/EcosystemReserve.sol#43-49) ignores return value by token.transfer(recipient,amount) (contracts/EcosystemReserve.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

EcosystemReserve.approve(IERC20,address,uint256) (contracts/EcosystemReserve.sol#35-41) ignores return value by token.approve(recipient,amount) (contracts/EcosystemReserve.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Address.isContract(address) (contracts/libraries/Address.sol#26-37) uses assembly
        - INLINE ASM (contracts/libraries/Address.sol#33-35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address.isContract(address) (contracts/libraries/Address.sol#26-37) is never used and should be removed
Address.sendValue(address,uint256) (contracts/libraries/Address.sol#55-61) is never used and should be removed
SafeERC20._callOptionalReturn(IERC20,bytes) (contracts/libraries/SafeERC20.sol#42-53) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (contracts/libraries/SafeERC20.sol#30-35) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (contracts/libraries/SafeERC20.sol#37-40) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (contracts/libraries/SafeERC20.sol#22-24) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (contracts/libraries/SafeERC20.sol#26-28) is never used and should be removed
SafeMath.add(uint256,uint256) (contracts/libraries/SafeMath.sol#28-33) is never used and should be removed
SafeMath.div(uint256,uint256) (contracts/libraries/SafeMath.sol#102-104) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/libraries/SafeMath.sol#117-128) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/libraries/SafeMath.sol#141-143) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/libraries/SafeMath.sol#156-163) is never used and should be removed
SafeMath.mul(uint256,uint256) (contracts/libraries/SafeMath.sol#77-89) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/libraries/SafeMath.sol#44-46) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (contracts/libraries/SafeMath.sol#57-66) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Low level call in Address.sendValue(address,uint256) (contracts/libraries/Address.sol#55-61):
        - (success) = recipient.call{value: amount}() (contracts/libraries/Address.sol#59)
Low level call in SafeERC20._callOptionalReturn(IERC20,bytes) (contracts/libraries/SafeERC20.sol#42-53):
        - (success,returndata) = address(token).call(data) (contracts/libraries/SafeERC20.sol#46)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable EcosystemReserve._fundsAdmin (contracts/EcosystemReserve.sol#18) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```



```
Initializable is re-used:
        - node_modules/@openzeppelin/upgrades-core/contracts/Initializable.sol#18-64
        - contracts/utils/Initializable.sol#12-42
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

DistributionManager._getAssetIndex(uint256,uint256,uint128,uint256) (contracts/DistributionManager.sol#212-235) uses a dangerous strict equality:
        - emissionPerSecond == 0 || totalBalance == 0 || lastUpdateTimestamp == block.timestamp || lastUpdateTimestamp >= DISTRIBUTION_END (contracts/DistributionManager.sol#219-222)
DistributionManager._updateAssetStateInternal(address,DistributionManager.AssetData,uint256) (contracts/DistributionManager.sol#73-100) uses a dangerous strict equality:
        - block.timestamp == lastUpdateTimestamp (contracts/DistributionManager.sol#81)
ERC20WithSnapshot._writeSnapshot(address,uint128,uint128) (contracts/libraries/ERC20WithSnapshot.sol#40-55) uses a dangerous strict equality:
        - ownerCountOfSnapshots != 0 && snapshotsOwner[ownerCountOfSnapshots.sub(1)].blockNumber == currentBlock (contracts/libraries/ERC20WithSnapshot.sol#47)
StakedToken.getNextCooldownTimestamp(uint256,uint256,address,uint256) (contracts/StakedToken.sol#251-285) uses a dangerous strict equality:
        - toCooldownTimestamp == 0 (contracts/StakedToken.sol#258)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in StakedToken.redeem(address,uint256) (contracts/StakedToken.sol#114-142):
        External calls:
        - _burn(msg.sender,amountToRedeem) (contracts/StakedToken.sol#133)
                - governance.onTransfer(from,to,amount) (contracts/libraries/ERC20WithSnapshot.sol#83)
        State variables written after the call(s):
        - stakersCooldowns[msg.sender] = 0 (contracts/StakedToken.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

StakedToken.__StakedToken_init(IERC20,IERC20,uint256,uint256,address,address,uint128,string,string,uint8,address).name (contracts/StakedToken.sol#52) shadows:
        - ERC20.name() (contracts/libraries/ERC20.sol#33-35) (function)
        - IERC20Detailed.name() (contracts/interfaces/IERC20Detailed.sol#10) (function)
StakedToken.__StakedToken_init(IERC20,IERC20,uint256,uint256,address,address,uint128,string,string,uint8,address).symbol (contracts/StakedToken.sol#53) shadows:
        - ERC20.symbol() (contracts/libraries/ERC20.sol#40-42) (function)
        - IERC20Detailed.symbol() (contracts/interfaces/IERC20Detailed.sol#11) (function)
StakedToken.__StakedToken_init(IERC20,IERC20,uint256,uint256,address,address,uint128,string,string,uint8,address).decimals (contracts/StakedToken.sol#54) shadows:
        - ERC20.decimals() (contracts/libraries/ERC20.sol#47-49) (function)
        - IERC20Detailed.decimals() (contracts/interfaces/IERC20Detailed.sol#12) (function)
ERC20.__ERC20_init_unchained(string,string,uint8).name (contracts/libraries/ERC20.sol#24) shadows:
        - ERC20.name() (contracts/libraries/ERC20.sol#33-35) (function)
        - IERC20Detailed.name() (contracts/interfaces/IERC20Detailed.sol#10) (function)
ERC20.__ERC20_init_unchained(string,string,uint8).symbol (contracts/libraries/ERC20.sol#24) shadows:
        - ERC20.symbol() (contracts/libraries/ERC20.sol#40-42) (function)
        - IERC20Detailed.symbol() (contracts/interfaces/IERC20Detailed.sol#11) (function)
ERC20.__ERC20_init_unchained(string,string,uint8).decimals (contracts/libraries/ERC20.sol#24) shadows:
        - ERC20.decimals() (contracts/libraries/ERC20.sol#47-49) (function)
        - IERC20Detailed.decimals() (contracts/interfaces/IERC20Detailed.sol#12) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Reentrancy in StakedToken.claimRewards(address,uint256) (contracts/StakedToken.sol#161-174):
        External calls:
        - IERC20(REWARD_TOKEN).safeTransferFrom(REWARDS_VAULT,to,amountToClaim) (contracts/StakedToken.sol#171)
        Event emitted after the call(s):
        - RewardsClaimed(msg.sender,to,amountToClaim) (contracts/StakedToken.sol#173)
```

As a result of the tests carried out with the Slither tool, some results were obtained and these results were reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives and these results were not included in the report. The actual vulnerabilities found by Slither are already included in the report findings.

THANK YOU FOR CHOOSING

HALBORN