// HALBORN

# Meld - Staking

## Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **October 23rd, 2023 - November 2nd, 2023**

Visit: **Halborn.com**

# DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION | DATE |
|---|---|---|
| 0.1 | Document Creation | 11/02/2023 |
| 0.2 | Draft Review | 11/02/2023 |
| 0.3 | Document Updates | 12/08/2023 |
| 1.0 | Remediation Plan | 12/14/2023 |
| 1.1 | Remediation Plan Review | 12/17/2023 |

# 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 |
| Ferran Celades | Halborn | Ferran.Celades@halborn.com |

# EXECUTIVE OVERVIEW

## 1.1 INTRODUCTION

Meld engaged Halborn to conduct a security assessment on their smart contract beginning on October 23rd, 2023 and ending on November 2nd, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

## 1.2 ASSESSMENT SUMMARY

Halborn was provided about one week for the engagement and assigned one full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

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

In summary, Halborn identified several vulnerabilities of varying severity in the smart contract code, which were mostly addressed by the Meld team.

During the assessment, the following components and functionalities were scrutinized:

**Contract Initialization and Configuration**
- Address provider initialization and immutability.
- Proper setting and role management of trusted forwarders.
- Withdrawal mechanics and the implication of deactivated nodes on it.

**Staking Mechanics**
- Staking and unstaking processes, including validation of lock tiers and

staking amounts.
- The accuracy of reward calculations and updates.

**Role-Based Access Control**
- Assignment, revocation, and renouncement of roles.
- Emergency stop functionality through pausing and unpausing of the contract.

**Token Handling**
- ERC20, ERC721, and ERC1155 token rescue functions, ensuring proper access control and restrictions.

**Smart Contract Libraries Usage**
- Use of libraries for managing node and staker data, ensuring they handle edge cases without errors.

**Upgrade Patterns**
- Potential for future upgrades and how address providers manage contract address changes.

**Reward Management**
- Mechanisms of reward accumulation and distribution, including handling of inactive nodes and slashing.

**Meta Transaction Handling**
- Implications of meta transactions on role-based functions and overall protocol interactions.

Questions and Scenarios for Consideration::

- Impact of node status changes mid-epoch on user actions and rewards.
- Interactions with the protocol via meta transactions and their implications.
- Treatment of rewards versus staked tokens in the event of slashing.
- Adequacy of price/meld checks post-node departure and withdrawal operations.
- Potential for increased token issuance due to specific loops and conditionals within the staking logic.

- User actions in the context of node deactivation, including staking, changing delegation, and claiming or updating rewards.
- Effects of contract interactions on the rewards and ledger balance post-node departure and during active epochs.

## 1.3 SCOPE

The assessment was scoped into the following smart contracts:

- contracts/MeldStakingStorage.sol
- contracts/MeldStakingCommon.sol
- contracts/MeldStakingNFTMetadata.sol
- contracts/MeldStakingConfig.sol
- contracts/MeldStakingDelegator.sol
- contracts/interfaces/IMeldStakingStorage.sol
- contracts/MeldStakingOperator.sol
- contracts/MeldStakingNFT.sol
- contracts/interfaces/IMeldStakingConfig.sol
- contracts/libraries/NodeLibrary.sol
- contracts/base/MeldStakingBase.sol
- contracts/MeldStakingAddressProvider.sol
- contracts/Errors.sol
- contracts/libraries/GlobalLibrary.sol
- contracts/interfaces/IMeldStakingCommon.sol
- contracts/interfaces/IMeldStakingNFT.sol
- contracts/interfaces/IMeldStakingAddressProvider.sol
- contracts/interfaces/IMeldStakingOperator.sol
- contracts/libraries/StakerLibrary.sol
- contracts/interfaces/IMeldStakingDelegator.sol
- contracts/base/RescueTokens.sol
- contracts/interfaces/IMeldStakingCommonEvents.sol
- contracts/interfaces/IMeldStakingNFTMetadata.sol

Commit ID: 783e6b91112cae42cb3f4234a04e841e97662f5b (audit branch)

Repository URL: https://github.com/MELD-labs/meld-evm-staking

## 1.4 TEST APPROACH & METHODOLOGY

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

- 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.
- Manual testing by custom scripts.
- Testnet deployment (Foundry).

# 2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

# 2.1 EXPLOITABILITY

### Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

### Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

### Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

### Metrics:

| Exploitability Metric ($m_E$) | Metric Value | Numerical Value |
|---|---|---|
| Attack Origin (AO) | Arbitrary (AO:A)<br>Specific (AO:S) | 1<br>0.2 |
| Attack Cost (AC) | Low (AC:L)<br>Medium (AC:M)<br>High (AC:H) | 1<br>0.67<br>0.33 |
| Attack Complexity (AX) | Low (AX:L)<br>Medium (AX:M)<br>High (AX:H) | 1<br>0.67<br>0.33 |

Exploitability $E$ is calculated using the following formula:

$$E = \prod m_e$$

## 2.2 IMPACT

### Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

### Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

### Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

### Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

### Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

| Impact Metric ($m_I$) | Metric Value | Numerical Value |
|---|---|---|
| Confidentiality (C) | None (I:N) | 0 |
| | Low (I:L) | 0.25 |
| | Medium (I:M) | 0.5 |
| | High (I:H) | 0.75 |
| | Critical (I:C) | 1 |
| Integrity (I) | None (I:N) | 0 |
| | Low (I:L) | 0.25 |
| | Medium (I:M) | 0.5 |
| | High (I:H) | 0.75 |
| | Critical (I:C) | 1 |
| Availability (A) | None (A:N) | 0 |
| | Low (A:L) | 0.25 |
| | Medium (A:M) | 0.5 |
| | High (A:H) | 0.75 |
| | Critical | 1 |
| Deposit (D) | None (D:N) | 0 |
| | Low (D:L) | 0.25 |
| | Medium (D:M) | 0.5 |
| | High (D:H) | 0.75 |
| | Critical (D:C) | 1 |
| Yield (Y) | None (Y:N) | 0 |
| | Low (Y:L) | 0.25 |
| | Medium: (Y:M) | 0.5 |
| | High: (Y:H) | 0.75 |
| | Critical (Y:H) | 1 |

Impact $I$ is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

## 2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

| Coefficient ($C$) | Coefficient Value | Numerical Value |
|---|---|---|
| Reversibility ($r$) | None (R:N)<br>Partial (R:P)<br>Full (R:F) | 1<br>0.5<br>0.25 |
| Scope ($s$) | Changed (S:C)<br>Unchanged (S:U) | 1.25<br>1 |

Severity Coefficient $C$ is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score $S$ is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

| Severity | Score Value Range |
|---|---|
| Critical | 9 - 10 |
| High | 7 - 8.9 |
| Medium | 4.5 - 6.9 |
| Low | 2 - 4.4 |
| Informational | 0 - 1.9 |

# 3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 5 | 2 | 0 | 2 | 5 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| (HAL-01) WITHDRAWALS FAIL ON DEACTIVATED NODES | Critical (10) | SOLVED - 11/08/2023 |
| (HAL-02) INFLATED STAKING AMOUNT POST-DELEGATION CHANGE | Critical (10) | SOLVED - 11/08/2023 |
| (HAL-03) ERRONEOUS FEE TRACKING ON NODE DELEGATION CHANGE | Critical (10) | SOLVED - 11/08/2023 |
| (HAL-04) WEIGHT DISCREPANCY ON DELEGATION CHANGE | Critical (10) | SOLVED - 11/08/2023 |
| (HAL-05) WEIGHTED AMOUNTS ARE NOT BEING REMOVED ON SLASHING | Critical (10) | SOLVED - 12/10/2023 |
| (HAL-06) DIVISION BY ZERO | High (8.8) | SOLVED - 11/20/2023 |
| (HAL-07) ARITHMETIC UNDERFLOW | High (8.8) | SOLVED - 08/11/2023 |
| (HAL-08) JSON AND SVG INJECTION | Low (3.1) | RISK ACCEPTED |
| (HAL-09) POTENTIAL TOKEN THEFT | Low (2.2) | SOLVED - 11/08/2023 |
| (HAL-10) REMOVE NODE DATA INCONSISTENCY | Informational (1.8) | SOLVED - 11/08/2023 |
| (HAL-11) MAXIMUM STAKING AMOUNT BELOW CURRENT STAKE | Informational (1.0) | NOT APPLICABLE |
| (HAL-12) INEFFICIENT GAS USAGE ON REMOVE TIER | Informational (0.5) | ACKNOWLEDGED |
| (HAL-13) IMPROPER INITIALIZATION CHECKS | Informational (1.5) | SOLVED - 11/08/2023 |
| (HAL-14) INCORRECT DOCUMENTATION | Informational (0.0) | SOLVED - 11/08/2023 |

# FINDINGS & TECH DETAILS

# 4.1 (HAL-01) WITHDRAWALS FAIL ON DEACTIVATED NODES - CRITICAL(10)

Description:

The withdraw function in the smart contract incorrectly uses the getCurrentEpoch function to determine if a staker can withdraw their funds. The subsequent call to updateStakerPreviousEpochs relies on this epoch check to determine the last active epoch of the node. If the node is deactivated, the function reverts with a INVALID_EPOCH error, thus preventing any withdrawal attempts. This behavior is inconsistent with the intended functionality, as it should be possible to withdraw from a deactivated node.

Proof of Concept:

**Listing 1**

```
    function test_halborn_leave_withdraw() external {
        vm.warp(block.timestamp + 2);

        _meldToken.mock_mint(USER1, 200_000 * 1e18);

        vm.startPrank(USER1);
        _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
        console.log("Requesting node NAME1 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME1")));
        // 5% fees
        _meldStakingOperator.requestNode('NAME1', 500, 100_000 * 1
↳ e18, 0, "");
        uint256 nodeStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();

        vm.stopPrank();

        vm.startPrank(ADMIN);
        _meldStakingConfig.approveNodeRequest(_meldStakingOperator
↳ .hashNodeId("NAME1"));
```

```
17          _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 15000);
18          vm.stopPrank();
19
20          _meldToken.mock_mint(USER2, 100_000 * 1e18);
21
22          vm.startPrank(USER2);
23          _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
24          _meldStakingDelegator.stake(100_000 * 1e18,
↳ _meldStakingOperator.hashNodeId("NAME1"), 1);
25          uint256 userStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();
26          vm.stopPrank();
27
28          vm.warp(block.timestamp + 10 days);
29
30          vm.startPrank(USER1);
31          _meldStakingOperator.leaveNode(nodeStakingNFTId);
32          vm.stopPrank();
33
34          // We wait some days for user2 to withdraw
35          vm.warp(block.timestamp + 15 days);
36
37          console.log("meldToken balance         (user1):",
↳ _meldToken.balanceOf(USER1)/ 1e18);
38          console.log("meldToken balance         (user2):",
↳ _meldToken.balanceOf(USER2)/ 1e18);
39
40          vm.startPrank(USER2);
41          // This will fail, as the withraw will be using
↳ getCurrentEpoch and use it for the _untilEpoch
42          // under updateStakerPreviousEpochs which will cause the `
↳ require(_untilEpoch <= untilEpoch, INVALID_EPOCH);`
43          // to fail
44          _meldStakingDelegator.withdraw(userStakingNFTId);
45          vm.stopPrank();
46
47          console.log("After withdraw");
48          console.log("meldToken balance         (user1):",
↳ _meldToken.balanceOf(USER1)/ 1e18);
49          console.log("meldToken balance         (user2):",
↳ _meldToken.balanceOf(USER2)/ 1e18);
50      }
```

Output:

**Listing 2**

```
[5666] MeldStakingCommon::updateStakerPreviousEpochs(2, 6)
   [578] MeldStakingStorage::getStakerNodeId(2) [staticcall]
      0 xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36
   [615] MeldStakingStorage::isNode(0 xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36) [staticcall]
      true
   [556] MeldStakingStorage::getNodeEndTimestamp(0 xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36) [staticcall]
      864003 [8.64e5]
   [1050] MeldStakingStorage::getEpoch(864003 [8.64e5]) [staticcall]
      3
    "MeldStaking: Invalid epoch"
  "MeldStaking: Invalid epoch"
"MeldStaking: Invalid epoch"
```

BVSS:

**AO:A/AC:L/AX:L/C:M/I:C/A:H/D:N/Y:M/R:N/S:U (10)**

Recommendation:

The smart contract should be updated to rectify the withdrawal issue for deactivated nodes. Instead of relying on `stakingStorage.getCurrentEpoch()` for the current epoch during the withdrawal process, the contract should utilize `_getLastActiveEpoch(nodeId)` to ascertain the last epoch in which the node was active. This change would allow for the correct determination of the staker's ability to withdraw their funds irrespective of the node's current activation status. The `_getLastActiveEpoch` method should return the most recent epoch during which the node was active, ensuring that the withdrawal process completes successfully even if the

node has been deactivated. This approach aligns with the expected behavior and allows users to retrieve their staked funds from nodes that are no longer active. Comprehensive testing must follow to verify that the withdrawal functionality operates correctly in all scenarios, including those involving deactivated nodes.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR26. The code uses getLastActiveEpoch in many places, including the one described in this issue.

## 4.2 (HAL-02) INFLATED STAKING AMOUNT POST-DELEGATION CHANGE - CRITICAL(10)

Description:

The changeDelegation function in the MeldStakingDelegator contract contains an inflation vulnerability due to incorrect fee calculations when users change their delegation from one node to another. The logic in the function currently inflates the amount of tokens that the user (NFT holder) has staked.

The specific line in question:

**Listing 3**

```
stakingStorage.setStakerLastStakedAmountPerEpoch(
    _nftId,
    currentEpoch,
    t.stakerBaseStakedAmount + t.oldFeeAmount - t.newFeeAmount
);
```

This line improperly adds the oldFeeAmount to the stakerBaseStakedAmount, even though stakerBaseStakedAmount is the base value without any fees subtracted. As a result, when changing delegation, the system erroneously believes the user has staked more tokens than they actually have, which can lead to disproportionate rewards distribution and potential token supply inflation.

Proof of Concept:

When a stake of 100,000 was made on node 5, which has a 5% fee, the expected value for getStakerLastStakedAmountPerEpoch should be 95,000. If the delegation is then shifted to node 1, which has a 1% fee, the correct adjusted value should be 109,000. However, the Proof of Concept displays an inaccurate amount of 104,000.

**Listing 4**

```
1
2     function test_halborn_change_delegation_max_inflation()
↳ external {
3
4         vm.warp(block.timestamp + 2);
5
6         _meldToken.mock_mint(USER1, 500_000 * 1e18);
7
8         vm.startPrank(USER1);
9         for (uint256 index = 1; index <= 5; index++) {
10
11            _meldToken.approve(address(_meldStakingNFT), 100_000 *
↳  1e18);
12            string memory _nodeName = string.concat("NAME",
↳ Strings.toString(index));
13            bytes32 _nodeId = _meldStakingOperator.hashNodeId(
↳ _nodeName);
14            console.log("Requesting node", _nodeName , "with id: "
↳ , bytes32ToLiteralString(_nodeId));
15            // 5% fees
16            _meldStakingOperator.requestNode(_nodeName, 100 *
↳ index, 100_000 * 1e18, 0, "");
17            uint256 nodeStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();
18
19        }
20        vm.stopPrank();
21
22        vm.startPrank(ADMIN);
23        for (uint256 index = 1; index <= 5; index++) {
24
25            _meldToken.approve(address(_meldStakingNFT), 100_000 *
↳  1e18);
26            string memory _nodeName = string.concat("NAME",
↳ Strings.toString(index));
27            bytes32 _nodeId = _meldStakingOperator.hashNodeId(
↳ _nodeName);
28            _meldStakingConfig.approveNodeRequest(_nodeId);
29
30        }
31        _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 15000);
```

```
32          _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 20000);
33          vm.stopPrank();
34
35          ////////////////////////////////
36
37          _meldToken.mock_mint(USER2, 100_000 * 1e18);
38
39          vm.startPrank(USER2);
40          _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
41          _meldStakingDelegator.stake(100_000 * 1e18,
↳ _meldStakingOperator.hashNodeId("NAME5"), 1);
42          uint256 userStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();
43
44          // _meldStakingDelegator.changeDelegation(userStakingNFTId
↳ , _meldStakingOperator.hashNodeId("NAME4"));
45          // _meldStakingDelegator.changeDelegation(userStakingNFTId
↳ , _meldStakingOperator.hashNodeId("NAME3"));
46          // _meldStakingDelegator.changeDelegation(userStakingNFTId
↳ , _meldStakingOperator.hashNodeId("NAME2"));
47          _meldStakingDelegator.changeDelegation(userStakingNFTId,
↳ _meldStakingOperator.hashNodeId("NAME1"));
48          vm.stopPrank();
49
50
51          for (uint256 index = 1; index <= _meldStakingStorage.
↳ getCurrentEpoch(); index++) {
52
53              console.log("============= EPOCH %s
↳ ==================", index);
54              console.log("getLastStakedAmountPerEpoch:
↳               ", _meldStakingStorage.getLastStakedAmountPerEpoch(
↳ index) / 1e18);
55              console.log("getMinStakedAmountPerEpoch:
↳                ", _meldStakingStorage.getMinStakedAmountPerEpoch(
↳ index) / 1e18);
56              console.log("getStakerLastStakedAmountPerEpoch (nft):
↳  ", _meldStakingStorage.getStakerLastStakedAmountPerEpoch(
↳ userStakingNFTId, index) / 1e18);
57              console.log("getStakerMinStakedAmountPerEpoch  (nft):
↳  ", _meldStakingStorage.getStakerMinStakedAmountPerEpoch(
↳ userStakingNFTId, index) / 1e18);
```

```
58              console.log("");
59
60          }
61
62      }
```

**Listing 5**

```
1   Requesting node NAME1 with id:  0
↳ xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36
2   Requesting node NAME2 with id:  0
↳ x2859e3f8d44bc1720f80693c9242d107e6adcb1202343d1a14aac49edf9ffc9e
3   Requesting node NAME3 with id:  0
↳ xa59a7011e513c90dd9dff63c522bd947e7c6dc854da69f6005f2279803b3a2c3
4   Requesting node NAME4 with id:  0
↳ xd05da445e8a11db829bba59640194770adcfc37baf97251e668cce6eadef8bcb
5   Requesting node NAME5 with id:  0
↳ x017bb843f9247f041761667595fbc311de884ddb8704e23da52dba580607a333
6   ============= EPOCH 1 ===================
7   getLastStakedAmountPerEpoch:               650000
8   getMinStakedAmountPerEpoch:                0
9   getStakerLastStakedAmountPerEpoch (nft):   104000 // This should
↳  be 149000
10  getStakerMinStakedAmountPerEpoch  (nft):   0
```

BVSS:

**AO:A/AC:L/AX:L/C:M/I:C/A:H/D:N/Y:M/R:N/S:U (10)**

Recommendation:

1. Update the changeDelegation function, especially the problematic line, to avoid adding the old fee to the base staked amount.
2. Implement thorough unit tests to verify the correct behavior of the changeDelegation function post-change.
3. Review other areas in the contract where similar fee calculations are being made to ensure there's no similar inflation vulnerability

elsewhere.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR27. The code is now using getStakerLastStakedAmountPerEpoch to fetch the weighted amount for that epoch on the delegation instead of using the base stake. This means that the user will not lose its weighted amount.

## 4.3 (HAL-03) ERRONEOUS FEE TRACKING ON NODE DELEGATION CHANGE - CRITICAL(10)

Description:

The MeldStakingDelegator contract provides a mechanism for users to change their delegation from one node to another. Typically, fees are only expected to be tracked under getStakerLastStakedAmountPerEpoch. However, after the execution of the changeDelegation function, fees are being incorrectly tracked under getNodeLastStakedAmountPerEpoch, which deviates from the expected behavior.

This inaccurate fee accounting has the potential to disrupt the reward distribution mechanism, particularly if getNodeLastStakedAmountPerEpoch is used in calculations or condition checks where fees shouldn't be considered.

Proof of Concept:

**Listing 6**

```
1
2         vm.warp(block.timestamp + 2);
3         _meldToken.mock_mint(USER1, 200_000 * 1e18);
4
5         vm.startPrank(USER1);
6         _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
7         console.log("Requesting node NAME1 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME1")));
8         // 5% fees
9         _meldStakingOperator.requestNode('NAME1', 500, 100_000 * 1
↳ e18, 0, "");
10        uint256 nodeStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();
11
12        _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
```

```
↳ e18);
13          console.log("Requesting node NAME2 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME2")));
14          // 1% fees
15          _meldStakingOperator.requestNode('NAME2', 100, 100_000 * 1
↳ e18, 0, "");
16          uint256 nodeStakingNFTId2 = _meldStakingNFT.
↳ getTotalMintedNfts();
17          vm.stopPrank();
18
19          vm.startPrank(ADMIN);
20          _meldStakingConfig.approveNodeRequest(_meldStakingOperator
↳ .hashNodeId("NAME1"));
21          _meldStakingConfig.approveNodeRequest(_meldStakingOperator
↳ .hashNodeId("NAME2"));
22          _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 15000);
23          _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 20000);
24          vm.stopPrank();
25
26          ////////////////////////////////
27
28          _meldToken.mock_mint(USER2, 100_000 * 1e18);
29
30          vm.startPrank(USER2);
31          _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
32          _meldStakingDelegator.stake(100_000 * 1e18,
↳ _meldStakingOperator.hashNodeId("NAME1"), 1);
33          uint256 userStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();
34
35          // DO A DELEGATION CHANGE TO NODE 2
36          _meldStakingDelegator.changeDelegation(userStakingNFTId,
↳ _meldStakingOperator.hashNodeId("NAME2"));
37          vm.stopPrank();
38
39          vm.warp(block.timestamp + 15 days);
40
41          console.log("getNodeBaseStakedAmount           (node):",
↳ _meldStakingStorage.getNodeBaseStakedAmount(_meldStakingOperator.
↳ hashNodeId("NAME1")) / 1e18);
42          console.log("getNodeBaseStakedAmount           (node2):",
```

```
↳ _meldStakingStorage.getNodeBaseStakedAmount(_meldStakingOperator.
↳ hashNodeId("NAME2")) / 1e18);
43          console.log("getStakerBaseStakedAmount         (node):",
↳ _meldStakingStorage.getStakerBaseStakedAmount(nodeStakingNFTId) /
↳ 1e18);
44          console.log("getStakerBaseStakedAmount         (node2):",
↳ _meldStakingStorage.getStakerBaseStakedAmount(nodeStakingNFTId2) /
↳  1e18);
45          console.log("getStakerBaseStakedAmount         (nft) :",
↳ _meldStakingStorage.getStakerBaseStakedAmount(userStakingNFTId) /
↳ 1e18);
46          console.log("getStakerUnclaimedRewards         (node):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(nodeStakingNFTId) /
↳ 1e18);
47          console.log("getStakerUnclaimedRewards         (node2):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(nodeStakingNFTId2) /
↳  1e18);
48          console.log("getStakerUnclaimedRewards          (nft):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(userStakingNFTId) /
↳ 1e18);
49
50          _meldStakingCommon.updateUnclaimedRewards(userStakingNFTId
↳ );
51          _meldStakingCommon.updateUnclaimedRewards(nodeStakingNFTId
↳ );
52          _meldStakingCommon.updateUnclaimedRewards(
↳ nodeStakingNFTId2);
53
54          for (uint256 index = 1; index <= _meldStakingStorage.
↳ getCurrentEpoch(); index++) {
55
56              console.log("============= EPOCH %s
↳ ==================", index);
57              console.log("getLastStakedAmountPerEpoch:
↳               ", _meldStakingStorage.getLastStakedAmountPerEpoch(
↳ index) / 1e18);
58              console.log("getMinStakedAmountPerEpoch:
↳                 ", _meldStakingStorage.getMinStakedAmountPerEpoch(
↳ index) / 1e18);
59              console.log("getNodeLastStakedAmountPerEpoch:  (node):
↳ ", _meldStakingStorage.getNodeLastStakedAmountPerEpoch(
↳ _meldStakingOperator.hashNodeId("NAME1"), index) / 1e18);
60              console.log("getNodeLastStakedAmountPerEpoch:  (node2)
↳ :", _meldStakingStorage.getNodeLastStakedAmountPerEpoch(
```

```
↳ _meldStakingOperator.hashNodeId("NAME2"), index) / 1e18);
61              console.log("getStakerLastStakedAmountPerEpoch (nft):
↳  ", _meldStakingStorage.getStakerLastStakedAmountPerEpoch(
↳ userStakingNFTId, index) / 1e18);
62              console.log("getStakerMinStakedAmountPerEpoch  (nft):
↳  ", _meldStakingStorage.getStakerMinStakedAmountPerEpoch(
↳ userStakingNFTId, index) / 1e18);
63              console.log("getStakerLastStakedAmountPerEpoch (node):
↳  ", _meldStakingStorage.getStakerLastStakedAmountPerEpoch(
↳ nodeStakingNFTId, index) / 1e18);
64              console.log("getStakerMinStakedAmountPerEpoch  (node):
↳  ", _meldStakingStorage.getStakerMinStakedAmountPerEpoch(
↳ nodeStakingNFTId, index) / 1e18);
65              console.log("getStakerLastStakedAmountPerEpoch (node2)
↳ :", _meldStakingStorage.getStakerLastStakedAmountPerEpoch(
↳ nodeStakingNFTId2, index) / 1e18);
66              console.log("getStakerMinStakedAmountPerEpoch  (node2)
↳ :", _meldStakingStorage.getStakerMinStakedAmountPerEpoch(
↳ nodeStakingNFTId2, index) / 1e18);
67              console.log("");
68
69          }
```

> The only difference between the erroneous and the normal output in the POC is to comment out the changeDelegation line. Only the first epoch is shown:

Normal output:

**Listing 7**

```
1   getNodeBaseStakedAmount           (node): 100000
2   getNodeBaseStakedAmount           (node2): 200000
3   getStakerBaseStakedAmount         (node): 100000
4   getStakerBaseStakedAmount         (node2): 100000
5   getStakerBaseStakedAmount         (nft) : 100000
6   getStakerUnclaimedRewards         (node): 0
7   getStakerUnclaimedRewards         (node2): 0
8   getStakerUnclaimedRewards          (nft): 0
9   ============= EPOCH 1 ===================
10  getLastStakedAmountPerEpoch:               350000
```

```
  getMinStakedAmountPerEpoch:                 0
  getNodeLastStakedAmountPerEpoch:  (node): 100000
  getNodeLastStakedAmountPerEpoch:  (node2): 250000
  getStakerLastStakedAmountPerEpoch (nft):   149000
  getStakerMinStakedAmountPerEpoch  (nft):   0
  getStakerLastStakedAmountPerEpoch (node):  100000
  getStakerMinStakedAmountPerEpoch  (node):  0
  getStakerLastStakedAmountPerEpoch (node2): 101000
  getStakerMinStakedAmountPerEpoch  (node2): 0
```

Erroneous output:

**Listing 8**

```
  getNodeBaseStakedAmount           (node): 100000
  getNodeBaseStakedAmount           (node2): 200000
  getStakerBaseStakedAmount         (node): 100000
  getStakerBaseStakedAmount         (node2): 100000
  getStakerBaseStakedAmount         (nft) : 100000
  getStakerUnclaimedRewards         (node): 0
  getStakerUnclaimedRewards         (node2): 0
  getStakerUnclaimedRewards          (nft): 0
  ============= EPOCH 1 ====================
  getLastStakedAmountPerEpoch:                350000
  getMinStakedAmountPerEpoch:                 0
  getNodeLastStakedAmountPerEpoch:  (node): 145000 // fees are
↳ tracked here
  getNodeLastStakedAmountPerEpoch:  (node2): 205000 // fees are
↳ tracked here
  getStakerLastStakedAmountPerEpoch (nft):   104000
  getStakerMinStakedAmountPerEpoch  (nft):   0
  getStakerLastStakedAmountPerEpoch (node):  100000 // fees are
↳ NOT ADDED here
  getStakerMinStakedAmountPerEpoch  (node):  0
  getStakerLastStakedAmountPerEpoch (node2): 101000 // fees are
↳ NOT removed here
  getStakerMinStakedAmountPerEpoch  (node2): 0
```

BVSS:

**AO:A/AC:L/AX:L/C:M/I:C/A:H/D:N/Y:M/R:N/S:U (10)**

Recommendation:

The changeDelegation function needs to be revised to ensure that it correctly updates the getNodeLastStakedAmountPerEpoch and getStakerLastStakedAmountPerEpoch without introducing the fee into the former. This requires careful consideration of how fees are added or deducted during the delegation change process.

Any change should be thoroughly tested to confirm that the issue is resolved and that there are no further unintended side effects. Additionally, a broader review of how fees are handled across the contract might be beneficial to ensure that similar issues aren't present elsewhere.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR28. To get the weighted amount without fees, the code is now using stakingCommon.getWeightedAmount.

## 4.4 (HAL-04) WEIGHT DISCREPANCY ON DELEGATION CHANGE - CRITICAL(10)

Description:

The MeldStakingDelegator contract allows users to change their delegation from one node to another. However, it appears that the weighted stake associated with the staker isn't correctly transferred from the old node to the new node during this process. This oversight leads to incorrect weight accounting, which directly impacts the reward distribution mechanism.

Furthermore, when the staking lock eventually expires, the system will deduct the staked amount associated with the old node, despite it having already been removed during the delegation change. This creates a discrepancy in the staked amounts.

Proof of Concept:

**Listing 9**

```
        vm.warp(block.timestamp + 2);

        _meldToken.mock_mint(USER1, 200_000 * 1e18);

        vm.startPrank(USER1);
        _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
        console.log("Requesting node NAME1 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME1")));
        // 5% fees
        _meldStakingOperator.requestNode('NAME1', 500, 100_000 * 1
↳ e18, 0, "");
        uint256 nodeStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();

        _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
        console.log("Requesting node NAME2 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME2")));
```

```
14          // 1% fees
15          _meldStakingOperator.requestNode('NAME2', 100, 100_000 * 1
↳ e18, 0, "");
16          uint256 nodeStakingNFTId2 = _meldStakingNFT.
↳ getTotalMintedNfts();
17          vm.stopPrank();
18
19          vm.startPrank(ADMIN);
20          _meldStakingConfig.approveNodeRequest(_meldStakingOperator
↳ .hashNodeId("NAME1"));
21          _meldStakingConfig.approveNodeRequest(_meldStakingOperator
↳ .hashNodeId("NAME2"));
22          _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 15000);
23          _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 20000);
24          vm.stopPrank();
25
26          ////////////////////////////////
27
28          _meldToken.mock_mint(USER2, 100_000 * 1e18);
29
30          vm.startPrank(USER2);
31          _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
32          _meldStakingDelegator.stake(100_000 * 1e18,
↳ _meldStakingOperator.hashNodeId("NAME1"), 1);
33          uint256 userStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();
34
35          // DO A DELEGATION CHANGE TO NODE 2
36          _meldStakingDelegator.changeDelegation(userStakingNFTId,
↳ _meldStakingOperator.hashNodeId("NAME2"));
37          vm.stopPrank();
38
39          vm.warp(block.timestamp + 15 days);
40
41          console.log("getNodeBaseStakedAmount           (node):",
↳ _meldStakingStorage.getNodeBaseStakedAmount(_meldStakingOperator.
↳ hashNodeId("NAME1")) / 1e18);
42          console.log("getNodeBaseStakedAmount           (node2):",
↳ _meldStakingStorage.getNodeBaseStakedAmount(_meldStakingOperator.
↳ hashNodeId("NAME2")) / 1e18);
43          console.log("getStakerBaseStakedAmount         (node):",
```

```
↳ _meldStakingStorage.getStakerBaseStakedAmount(nodeStakingNFTId) /
↳ 1e18);
44          console.log("getStakerBaseStakedAmount         (node2):",
↳ _meldStakingStorage.getStakerBaseStakedAmount(nodeStakingNFTId2) /
↳  1e18);
45          console.log("getStakerBaseStakedAmount         (nft) :",
↳ _meldStakingStorage.getStakerBaseStakedAmount(userStakingNFTId) /
↳ 1e18);
46          console.log("getStakerUnclaimedRewards         (node):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(nodeStakingNFTId) /
↳ 1e18);
47          console.log("getStakerUnclaimedRewards         (node2):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(nodeStakingNFTId2) /
↳  1e18);
48          console.log("getStakerUnclaimedRewards          (nft):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(userStakingNFTId) /
↳ 1e18);
49
50          _meldStakingCommon.updateUnclaimedRewards(userStakingNFTId
↳ );
51          _meldStakingCommon.updateUnclaimedRewards(nodeStakingNFTId
↳ );
52          _meldStakingCommon.updateUnclaimedRewards(
↳ nodeStakingNFTId2);
53
54          for (uint256 index = 1; index <= _meldStakingStorage.
↳ getCurrentEpoch(); index++) {
55
56              console.log("============= EPOCH %s
↳ ====================", index);
57              console.log("getLastStakedAmountPerEpoch:
↳               ", _meldStakingStorage.getLastStakedAmountPerEpoch(
↳ index) / 1e18);
58              console.log("getMinStakedAmountPerEpoch:
↳                ", _meldStakingStorage.getMinStakedAmountPerEpoch(
↳ index) / 1e18);
59              console.log("getNodeLastStakedAmountPerEpoch:  (node):
↳ ", _meldStakingStorage.getNodeLastStakedAmountPerEpoch(
↳ _meldStakingOperator.hashNodeId("NAME1"), index) / 1e18);
60              console.log("getNodeLastStakedAmountPerEpoch:  (node2)
↳ :", _meldStakingStorage.getNodeLastStakedAmountPerEpoch(
↳ _meldStakingOperator.hashNodeId("NAME2"), index) / 1e18);
61              console.log("getStakerLastStakedAmountPerEpoch (nft):
↳  ", _meldStakingStorage.getStakerLastStakedAmountPerEpoch(
```

```
↳ userStakingNFTId, index) / 1e18);
62              console.log("getStakerMinStakedAmountPerEpoch  (nft):
↳  ", _meldStakingStorage.getStakerMinStakedAmountPerEpoch(
↳ userStakingNFTId, index) / 1e18);
63              console.log("getStakerLastStakedAmountPerEpoch (node):
↳  ", _meldStakingStorage.getStakerLastStakedAmountPerEpoch(
↳ nodeStakingNFTId, index) / 1e18);
64              console.log("getStakerMinStakedAmountPerEpoch  (node):
↳  ", _meldStakingStorage.getStakerMinStakedAmountPerEpoch(
↳ nodeStakingNFTId, index) / 1e18);
65              console.log("getStakerLastStakedAmountPerEpoch (node2)
↳ :", _meldStakingStorage.getStakerLastStakedAmountPerEpoch(
↳ nodeStakingNFTId2, index) / 1e18);
66              console.log("getStakerMinStakedAmountPerEpoch  (node2)
↳ :", _meldStakingStorage.getStakerMinStakedAmountPerEpoch(
↳ nodeStakingNFTId2, index) / 1e18);
67              console.log("");
68
69          }
```

unlock is shown):

**Listing 10**

```
1   getNodeBaseStakedAmount           (node): 100000
2   getNodeBaseStakedAmount           (node2): 200000
3   getStakerBaseStakedAmount         (node): 100000
4   getStakerBaseStakedAmount         (node2): 100000
5   getStakerBaseStakedAmount         (nft) : 100000
6   getStakerUnclaimedRewards         (node): 0
7   getStakerUnclaimedRewards         (node2): 0
8   getStakerUnclaimedRewards          (nft): 0
9   ============= EPOCH 1 ==================
10  getLastStakedAmountPerEpoch:                350000
11  getMinStakedAmountPerEpoch:                 0
12  getNodeLastStakedAmountPerEpoch:  (node): 145000 // This should
↳ NOT have the weight.
13  getNodeLastStakedAmountPerEpoch:  (node2): 205000 // This SHOULD
↳  have the new weight.
14  getStakerLastStakedAmountPerEpoch (nft):   104000
15  getStakerMinStakedAmountPerEpoch  (nft):   0
16  getStakerLastStakedAmountPerEpoch (node):  100000
17  getStakerMinStakedAmountPerEpoch  (node):  0
```

```
  getStakerLastStakedAmountPerEpoch (node2): 101000
  getStakerMinStakedAmountPerEpoch  (node2): 0
...

  ============= EPOCH 4 ==================
  getLastStakedAmountPerEpoch:               300000
  getMinStakedAmountPerEpoch:                300000
  getNodeLastStakedAmountPerEpoch:  (node): 95000 // This should
↳ NOT have the weight substracted.
  getNodeLastStakedAmountPerEpoch:  (node2): 205000 // This SHOULD
↳  have the new weight substracted.
  getStakerLastStakedAmountPerEpoch (nft):   54000
  getStakerMinStakedAmountPerEpoch  (nft):   54000
  getStakerLastStakedAmountPerEpoch (node):  100000
  getStakerMinStakedAmountPerEpoch  (node):  100000
  getStakerLastStakedAmountPerEpoch (node2): 101000
  getStakerMinStakedAmountPerEpoch  (node2): 101000

```

BVSS:

**AO:A/AC:L/AX:L/C:M/I:C/A:H/D:N/Y:M/R:N/S:U (10)**

Recommendation:

To resolve the issue, during the changeDelegation process:

1. Compute the weighted stake that needs to be transferred from the old node to the new node.
2. Subtract this weighted stake from the old node.
3. Add this weighted stake to the new node.
4. Reflect these changes in any global state or variables that track total weighted stakes.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR29. The code is not using a custom created function named transferExcessWeight under common. This

function will transfer the excess weight from the old node to the new node only if the lock position is not liquid. If the issue “ARITHMETIC UNDERFLOW” is solved and since the `updateUnclaimedRewards` is called before the `transferExcessWeight` call, the `lockTierId` will reflect the latest value and those make sure that weights are not transferred even if delegation change is requested on the same epoch as when the position turns liquid.

# 4.5 (HAL-05) WEIGHTED AMOUNTS ARE NOT BEING REMOVED ON SLASHING - CRITICAL(10)

Description:

The issue arises from a discrepancy between the slashing of nodes and the handling of the associated NFTs and weighted global staking amounts. When a node is slashed, the associated MELD tokens are removed according to the slashing logic, but the NFTs remain so that users can claim their rewards up to the point of deactivation or slashing.

The main problem occurs with the global weighted amounts, which are not being removed immediately upon slashing, but rather wait until the user's staking transforms to liquid in the future epoch. If the slashed node was the only staking position in the system, or if the remaining staked amounts are less than the weights pending removal, this leads to an underflow when adjusting the global totals. In scenarios where there are other stakes in the system, the subtraction of the weighted amounts upon transformation to liquid still leads to incorrect global totals because the base stake amount has already been subtracted during slashing, but the weighted amounts are deferred until liquid transformation.

Proof of Concept:

**Listing 11**

```
    vm.warp(block.timestamp + 2);

    _meldToken.mock_mint(USER1, 100_000 * 1e18);

    vm.startPrank(USER1);
    _meldToken.approve(address(_meldStakingNFT), 100_000 * 1e18);
    console.log("Requesting node NAME1 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME1")));
    // 5% fees
    _meldStakingOperator.requestNode('NAME1', 500, 100_000 * 1e18,
```

```
↳  0, "");
10      uint256 nodeStakingNFTId = _meldStakingNFT.getTotalMintedNfts
↳ ();
11
12
13      vm.startPrank(ADMIN);
14      _meldStakingConfig.approveNodeRequest(_meldStakingOperator.
↳ hashNodeId("NAME1"));
15      _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 15000);
16      vm.stopPrank();
17
18      ////////////////////////////////
19
20      _meldToken.mock_mint(USER2, 100_000 * 1e18);
21
22      vm.startPrank(USER2);
23      _meldToken.approve(address(_meldStakingNFT), 100_000 * 1e18);
24      _meldStakingDelegator.stake(100_000 * 1e18,
↳ _meldStakingOperator.hashNodeId("NAME1"), 1);
25      uint256 userStakingNFTId = _meldStakingNFT.getTotalMintedNfts
↳ ();
26      vm.stopPrank();
27
28      vm.warp(block.timestamp + 10 days);
29
30      vm.startPrank(ADMIN);
31      _meldStakingConfig.slashNode(_meldStakingOperator.hashNodeId("
↳ NAME1"));
32      vm.stopPrank();
33
34      vm.warp(block.timestamp + 30 days);
35
36      console.log("getNodeBaseStakedAmount          (node):",
↳ _meldStakingStorage.getNodeBaseStakedAmount(_meldStakingOperator.
↳ hashNodeId("NAME1")) / 1e18);
37      console.log("getStakerBaseStakedAmount        (node):",
↳ _meldStakingStorage.getStakerBaseStakedAmount(nodeStakingNFTId) /
↳ 1e18);
38      console.log("getStakerBaseStakedAmount        (nft) :",
↳ _meldStakingStorage.getStakerBaseStakedAmount(userStakingNFTId) /
↳ 1e18);
39      console.log("getStakerUnclaimedRewards        (node):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(nodeStakingNFTId) /
```

```
↳ 1e18);
40
41      vm.startPrank(ADMIN);
42      _meldToken.approve(address(_meldStakingNFT), 100_000 * 1e18);
43      _meldToken.mock_mint(ADMIN, 100_000 * 1e18);
44      _meldStakingConfig.grantRole(_meldStakingConfig.
↳ REWARDS_SETTER_ROLE(), ADMIN);
45      _meldStakingConfig.setRewards(100 * 1e18, 2);
46      _meldStakingConfig.setRewards(100 * 1e18, 3);
47      _meldStakingConfig.setRewards(100 * 1e18, 4);
48      _meldStakingConfig.setRewards(100 * 1e18, 4);
49      vm.stopPrank();
50      console.log("getStakerUnclaimedRewards          (nft):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(userStakingNFTId) /
↳ 1e18);
51
52
53      _meldStakingCommon.updateUnclaimedRewards(userStakingNFTId);
54      _meldStakingCommon.updateUnclaimedRewards(nodeStakingNFTId);
```

**Listing 12**

```
1
2       [28102] MeldStakingConfig::setRewards(10000000000000000000
↳ [1e20], 4)
3           [426] MeldStakingStorage::getLastEpochRewardsUpdated() [
↳ staticcall]
4               3
5           [1055] MeldStakingStorage::getCurrentEpoch() [staticcall]
6               9
7           [23678] MeldStakingStorage::setRewards(4,
↳ 10000000000000000000 [1e20])
8               "Arithmetic over/underflow"
9            "Arithmetic over/underflow"
10         "Arithmetic over/underflow"
```

BVSS:

**AO:A/AC:L/AX:L/C:M/I:C/A:H/D:N/Y:M/R:N/S:U (10)**

Recommendation:

It is recommended to keep track of all weighted epochs for a node and have those be removed from the node state and more importantly from the global state upon slashing.

Remediation Plan:

**SOLVED**: The Meld team solved this issue on previous commits but merged with https://github.com/MELD-labs/meld-evm-staking/tree/feat/new-partial-slashing. This new branch does add the possibility to slash a partial amount from delegators. The node operator will have the 100% slashed but delegators will be able to withdraw the remaining amount from their positions.

## 4.6 (HAL-06) DIVISION BY ZERO - HIGH (8.8)

Description:

The problem stems from a division by zero occurring in the updateUnclaimedRewards function. This division by zero is triggered when the function attempts to calculate the rewards for epochs in which the node was slashed, particularly for epoch 3 in your scenario. When a node is slashed, it's possible for the getMinStakedAmountPerEpoch function to return 0 for that epoch, which is what is happening here.

The rewards are calculated based on the proportion of the staker's stake relative to the total minimum stake for that epoch. However, if the total minimum stake is zero (which can occur if all nodes are slashed or if there was an error in how the stakes were recorded), the function attempts to divide by zero, leading to a revert.

This situation can arise when staking systems allow for retroactive reward calculations that include past epochs where conditions might have changed drastically, such as nodes being slashed. This design may not have fully accounted for how slashes affect the calculation of past rewards.

Proof of Concept:

**Listing 13**

```
    vm.warp(block.timestamp + 2);

    _meldToken.mock_mint(USER1, 100_000 * 1e18);

    vm.startPrank(USER1);
    _meldToken.approve(address(_meldStakingNFT), 100_000 * 1e18);
    console.log("Requesting node NAME1 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME1")));
    // 5% fees
    _meldStakingOperator.requestNode('NAME1', 500, 100_000 * 1e18,
↳  0, "");
```

```
10      uint256 nodeStakingNFTId = _meldStakingNFT.getTotalMintedNfts
↳ ();
11
12
13      vm.startPrank(ADMIN);
14      _meldStakingConfig.approveNodeRequest(_meldStakingOperator.
↳ hashNodeId("NAME1"));
15      _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 2,
↳ 15000);
16      vm.stopPrank();
17
18      ////////////////////////////////
19
20      _meldToken.mock_mint(USER2, 100_000 * 1e18);
21
22      vm.startPrank(USER2);
23      _meldToken.approve(address(_meldStakingNFT), 100_000 * 1e18);
24      _meldStakingDelegator.stake(100_000 * 1e18,
↳ _meldStakingOperator.hashNodeId("NAME1"), 1);
25      uint256 userStakingNFTId = _meldStakingNFT.getTotalMintedNfts
↳ ();
26      vm.stopPrank();
27
28      vm.warp(block.timestamp + 10 days);
29
30      vm.startPrank(ADMIN);
31      _meldStakingConfig.slashNode(_meldStakingOperator.hashNodeId("
↳ NAME1"));
32      vm.stopPrank();
33
34      vm.warp(block.timestamp + 30 days);
35
36      console.log("getNodeBaseStakedAmount           (node):",
↳ _meldStakingStorage.getNodeBaseStakedAmount(_meldStakingOperator.
↳ hashNodeId("NAME1")) / 1e18);
37      console.log("getStakerBaseStakedAmount         (node):",
↳ _meldStakingStorage.getStakerBaseStakedAmount(nodeStakingNFTId) /
↳ 1e18);
38      console.log("getStakerBaseStakedAmount         (nft) :",
↳ _meldStakingStorage.getStakerBaseStakedAmount(userStakingNFTId) /
↳ 1e18);
39      console.log("getStakerUnclaimedRewards         (node):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(nodeStakingNFTId) /
↳ 1e18);
```

```
40
41      vm.startPrank(ADMIN);
42      _meldToken.approve(address(_meldStakingNFT), 100_000 * 1e18);
43      _meldToken.mock_mint(ADMIN, 100_000 * 1e18);
44      _meldStakingConfig.grantRole(_meldStakingConfig.
↳ REWARDS_SETTER_ROLE(), ADMIN);
45      _meldStakingConfig.setRewards(100 * 1e18, 2);
46      _meldStakingConfig.setRewards(100 * 1e18, 3);
47      vm.stopPrank();
48      console.log("getStakerUnclaimedRewards          (nft):",
↳ _meldStakingStorage.getStakerUnclaimedRewards(userStakingNFTId) /
↳ 1e18);
49
50
51      _meldStakingCommon.updateUnclaimedRewards(userStakingNFTId);
52      _meldStakingCommon.updateUnclaimedRewards(nodeStakingNFTId);
```

**Listing 14**

```
1
2           [698] MeldStakingStorage::getStakerMinStakedAmountPerEpoch
↳ (1, 2) [staticcall]
3               1050000000000000000000 [1.05e23]
4           [527] MeldStakingStorage::getMinStakedAmountPerEpoch(3) [
↳ staticcall]
5               0
6           [551] MeldStakingStorage::getTotalRewardsPerEpoch(3) [
↳ staticcall]
7               1000000000000000000 [1e20]
8           [698] MeldStakingStorage::getStakerMinStakedAmountPerEpoch
↳ (1, 3) [staticcall]
9               1050000000000000000000 [1.05e23]
10           "Division or modulo by 0"
11       "Division or modulo by 0"
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:H/A:M/D:N/Y:N/R:N/S:U (8.8)**

Recommendation:

To resolve this issue and prevent the division by zero error, the smart contract should be modified to handle cases where `getMinStakedAmountPerEpoch` returns `0`. This can include conditional checks that bypass the reward calculation for an epoch if the minimum staked amount is zero, possibly defaulting the reward to zero or some other appropriate value for that epoch.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in https://github.com/MELD-labs/meld-evm-staking/tree/bug/unclaimed-rewards.

## 4.7 (HAL-07) ARITHMETIC UNDERFLOW - HIGH (8.8)

Description:

In the changeDelegation function of the MeldStakingDelegator contract, the staker's delegation can change from one node operator to another. When this happens, a series of updates to different states in the staking system is necessary to ensure accuracy in reward distribution, among other things.

The issue pointed out is that while previous epochs for the staker, the new node, and the old node are being updated, the previous epochs for the operators associated with these nodes aren't being updated. This can lead to potential discrepancies when calculating reward shares, especially when an arithmetic underflow occurs if getStakerLastStakedAmountPerEpoch returns a value smaller than t.oldFeeAmount.

Proof of Concept:

**Listing 15**

```
        vm.warp(block.timestamp + 2);

        _meldToken.mock_mint(USER1, 200_000 * 1e18);

        vm.startPrank(USER1);
        _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
        console.log("Requesting node NAME1 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME1")));
        // 1% fees
        _meldStakingOperator.requestNode('NAME1', 100, 100_000 * 1
↳ e18, 0, "");
        uint256 nodeStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();

        _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
```

```
13          console.log("Requesting node NAME2 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME2")));
14          // 2% fees
15          _meldStakingOperator.requestNode('NAME2', 200, 100_000 * 1
↳ e18, 0, "");
16          uint256 nodeStakingNFTId2 = _meldStakingNFT.
↳ getTotalMintedNfts();
17          vm.stopPrank();
18
19          vm.startPrank(ADMIN);
20          _meldStakingConfig.approveNodeRequest(_meldStakingOperator
↳ .hashNodeId("NAME1"));
21          _meldStakingConfig.approveNodeRequest(_meldStakingOperator
↳ .hashNodeId("NAME2"));
22          _meldStakingConfig.addStakingLockTier(100_000 * 1e18, 4,
↳ 15000);
23          vm.stopPrank();
24
25          ////////////////////////////////
26
27          _meldToken.mock_mint(USER2, 100_000 * 1e18);
28
29          vm.startPrank(USER2);
30          _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
31          _meldStakingDelegator.stake(100_000 * 1e18,
↳ _meldStakingOperator.hashNodeId("NAME1"), 1);
32          uint256 userStakingNFTId = _meldStakingNFT.
↳ getTotalMintedNfts();
33          vm.stopPrank();
34
35          console.log("getStakerBaseStakedAmount         (nft): ",
↳ _meldStakingStorage.getStakerBaseStakedAmount(userStakingNFTId) /
↳ 1e18);
36
37          vm.warp(block.timestamp + 10 days);
38
39
40          vm.startPrank(USER2);
41          MeldStakingDelegator.TempData memory tmp =
↳ _meldStakingDelegator._getTempData(userStakingNFTId,
↳ _meldStakingOperator.hashNodeId("NAME1"), _meldStakingOperator.
↳ hashNodeId("NAME2"));
42          _meldStakingDelegator.changeDelegation(userStakingNFTId,
```

```
↳ _meldStakingOperator.hashNodeId("NAME2"));
43          vm.stopPrank();
```

**Listing 16**

```
1
2           [534] MeldStakingStorage::getStakerBaseStakedAmount(3) [
↳ staticcall]
3               100000000000000000000 [1e23]
4           [805] MeldStakingStorage::calculateDelegationFeeAmount(0
↳ xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36,
↳  100000000000000000000 [1e23]) [staticcall]
5               10000000000000000000 [1e21]
6           [805] MeldStakingStorage::calculateDelegationFeeAmount(0
↳ x2859e3f8d44bc1720f80693c9242d107e6adcb1202343d1a14aac49edf9ffc9e,
↳  100000000000000000000 [1e23]) [staticcall]
7               20000000000000000000 [2e21]
8           [1670] MeldStakingStorage::
↳ setStakerLastStakedAmountPerEpoch(3, 5, 990000000000000000000
↳ [9.9e22])
9               ()
10          [555] MeldStakingStorage::getNodeOperator(0
↳ xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36)
↳  [staticcall]
11              1
12          [2678] MeldStakingStorage::
↳ getStakerLastStakedAmountPerEpoch(1, 5) [staticcall]
13              0
14           "Arithmetic over/underflow"
15        "Arithmetic over/underflow"
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:H/A:M/D:N/Y:N/R:N/S:U (8.8)**

Recommendation:

To address this issue, you should ensure that the previous epochs for both old and new node operators are updated accordingly. You can do this by adding the following lines to the changeDelegation function:

**Listing 17**

```
uint256 newOperator = stakingStorage.getNodeOperator(_newNodeId);
stakingCommon.updateStakerPreviousEpochs(oldOperator, currentEpoch
↳ );
stakingCommon.updateStakerPreviousEpochs(newOperator, currentEpoch
↳ );
```

Make sure to place these lines after the old and new nodes' previous epochs have been updated and before the reward and staking amounts are calculated. This will ensure that all necessary states are updated in the correct sequence, avoiding any potential underflows or discrepancies.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR30. The code is now updating both the old and new nodes to the latest epoch before performing weight and fee transfers.

# 4.8 (HAL-08) JSON AND SVG INJECTION - LOW (3.1)

Description:

In the MeldStakingNFTMetadata contract, the name parameter of the requestNode function is directly incorporated into JSON and SVG outputs, potentially leading to an injection attack.

An attacker could submit malicious input in the name parameter of the requestNode function, as demonstrated:

**Listing 18**

```
_meldStakingOperator.requestNode('NAME","my_tag":"haborn"}], "
↳ external_url": "https://halborn.com" ,"unused":[{"extra": "empty',
↳  10, 100_000 * 1e18, 0, "");
```

This results in a malformed JSON output with injected attributes. Such a vulnerability can lead to potential misinformation, unexpected behavior, and other security risks on platforms that consume this JSON. Additionally, SVG injections can lead to a variety of attacks, including Cross-Site Scripting (XSS) if rendered by browsers.

BVSS:

**AO:A/AC:L/AX:L/C:L/I:L/A:N/D:N/Y:N/R:N/S:U (3.1)**

Recommendation:

1. **Input Validation**: Ensure all input values are strictly validated before use. Special characters that can result in a malformed JSON or SVG output should be filtered or escaped.

For JSON:

**Listing 19**

```
function sanitizeForJSON(string memory input) public pure returns
↳ (string memory) {
    // Replace or escape characters like ", {, }, [ and ].
    // This is a simplification; you'll need a robust
↳ implementation.
    return input;
}

```

For SVG:

**Listing 20**

```
function sanitizeForSVG(string memory input) public pure returns (
↳ string memory) {
    // Replace or escape characters that can have special meaning
↳ in SVG.
    // This is a simplification; you'll need a robust
↳ implementation.
    return input;
}
```

2. **Off-Chain Validation**: All node requests should be carefully evaluated off-chain. Implement checks for invalid characters in the name and other attributes. Reject or flag suspicious or malformed requests for manual review.

3. **Limit Use of Dynamic Data**: Wherever possible, minimize the direct incorporation of user-supplied data into outputs, especially formats like JSON and SVG.

Remediation Plan:

**RISK ACCEPTED**: The Meld team will be verifying data off-chain, as doing it on-chain is too costly. They will be taking care and extra precaution on the node request data validation.

# 4.9 (HAL-09) POTENTIAL TOKEN THEFT - LOW (2.2)

Description:

In the MeldStakingNFT contract, the rescueERC20 function allows an admin with the DEFAULT_ADMIN_ROLE to transfer any ERC20 tokens held by the contract to a specified address. This poses a significant risk, particularly in the context of the mint and _depositMeld functions, where users can deposit MELD tokens into the contract which increases the lockedMeldTokens balance. The current design permits the admin to rescue these locked tokens, essentially granting unrestricted access to all tokens held by the contract.

Proof of Concept:

**Listing 21**

```
    function test_halborn_can_rescue_melds() public {

        StakingAddressProviderMock _addressProvider = new
↳ StakingAddressProviderMock();

        _addressProvider.set_meldStakingCommon(ADMIN);
        _addressProvider.set_meldToken(address(meld));

        vm.prank(ADMIN);
        stakingNft.initialize(address(_addressProvider));

        meld.mock_mint(USER1, 100 * 1e18);

        vm.prank(USER1);
        meld.approve(address(stakingNft), 100 * 1e18);

        console.log("Before mint, stakingNft MELD balance:", meld.
↳ balanceOf(address(stakingNft)) / 1e18);

        // Now admin acts as a valid "staking common"
        vm.prank(ADMIN);
        stakingNft.mint(USER1, 100 * 1e18);
```

```
21
22          console.log("After mint, stakingNft MELD balance:", meld.
↳ balanceOf(address(stakingNft)) / 1e18);
23
24          console.log("Admin does call rescueERC20 on the MELD");
25          vm.prank(ADMIN);
26          stakingNft.rescueERC20(address(meld), ADMIN);
27
28          console.log("After rescueERC20, stakingNft MELD balance:",
↳  meld.balanceOf(address(stakingNft)) / 1e18);
29
30      }
```

BVSS:

**AO:S/AC:L/AX:L/C:H/I:H/A:H/D:N/Y:N/R:N/S:U (2.2)**

Recommendation:

Modify the rescueERC20 function to prevent the extraction of the lockedMeldTokens balance for the specific MELD token:

**Listing 22**

```
1 function rescueERC20(address _token, address _to, uint256 _amount)
↳  public onlyRole(DEFAULT_ADMIN_ROLE) {
2     require(_token != address(meldToken) || _amount <= meldToken.
↳ balanceOf(address(this)) - lockedMeldTokens, "Rescue exceeds
↳ allowed amount");
3     IERC20(_token).safeTransfer(_to, _amount);
4 }
```

1. Add an _amount parameter to specify the amount of tokens to rescue.
2. For the MELD token specifically, ensure that the amount being rescued does not exceed the difference between the total MELD token balance and the lockedMeldTokens.
3. For other tokens, they can be rescued without restrictions as they are not tracked in the lockedMeldTokens.

This approach secures the lockedMeldTokens from administrative extraction while still allowing for the rescue of other accidentally sent tokens.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR31. The rescueERC20 function prevents the meldToken token from being used. This enforces the operators to use rescueMeldTokens which does implement what was described as a recommendation.

## 4.10 (HAL-10) REMOVE NODE DATA INCONSISTENCY - INFORMATIONAL (1.8)

Description:

In the MeldStakingStorage contract, the function createNodeRequest is responsible for creating new node requests. It correctly populates the nodeRequests and nodeRequestsPerOperator mappings and adds the _nodeId to the activeNodeRequestsIds array. However, it fails to update the activeNodeRequestsIdsIndex mapping to track the index of the newly added _nodeId.

On the other hand, the function removeNodeRequest relies on the activeNodeRequestsIdsIndex mapping to fetch the index of the _nodeId to be removed from the activeNodeRequestsIds array. If the index mapping is not set or if it defaults to zero (since uninitialized mappings return 0 in Solidity), it will always attempt to remove the first element, leading to data inconsistency.

Proof of Concept:

**Listing 23**

```
    function test_halborn_invalid_remove_node() public {

        vm.warp(block.timestamp + 1 days);

        _meldToken.mock_mint(USER1, 100_000 * 1e18);
        _meldToken.mock_mint(USER2, 100_000 * 1e18);
        _meldToken.mock_mint(USER3, 100_000 * 1e18);

        vm.startPrank(USER1);
        _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
        console.log("Requesting node NAME1 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME1")));
        _meldStakingOperator.requestNode('NAME1', 10, 100_000 * 1
↳ e18, 0, "");
        vm.stopPrank();
```

```
14
15          vm.startPrank(USER2);
16          _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
17          console.log("Requesting node NAME2 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME2")));
18          _meldStakingOperator.requestNode('NAME2', 10, 100_000 * 1
↳ e18, 0, "");
19          vm.stopPrank();
20
21          vm.startPrank(USER3);
22          _meldToken.approve(address(_meldStakingNFT), 100_000 * 1
↳ e18);
23          console.log("Requesting node NAME3 with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME3")));
24          _meldStakingOperator.requestNode('NAME3', 10, 100_000 * 1
↳ e18, 0, "");
25          vm.stopPrank();
26
27
28          console.log("Removing NAME2 node with id: ",
↳ bytes32ToLiteralString(_meldStakingOperator.hashNodeId("NAME2")));
29          vm.startPrank(USER2);
30          _meldStakingOperator.cancelNodeRequest(
↳ _meldStakingOperator.hashNodeId("NAME2"));
31          vm.stopPrank();
32
33          console.log("We should have: ");
34          console.log(bytes32ToLiteralString(_meldStakingOperator.
↳ hashNodeId("NAME1")));
35          console.log(bytes32ToLiteralString(_meldStakingOperator.
↳ hashNodeId("NAME3")));
36
37          console.log("But we have: ");
38          console.log(bytes32ToLiteralString(_meldStakingStorage.
↳ activeNodeRequestsIds(0)));
39          console.log(bytes32ToLiteralString(_meldStakingStorage.
↳ activeNodeRequestsIds(1)));
40
41      }
```

**Listing 24**

```
  Requesting node NAME1 with id:  0
↳ xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36
  Requesting node NAME2 with id:  0
↳ x2859e3f8d44bc1720f80693c9242d107e6adcb1202343d1a14aac49edf9ffc9e
  Requesting node NAME3 with id:  0
↳ xa59a7011e513c90dd9dff63c522bd947e7c6dc854da69f6005f2279803b3a2c3
  Removing NAME2 node with id:  0
↳ x2859e3f8d44bc1720f80693c9242d107e6adcb1202343d1a14aac49edf9ffc9e
  We should have:
  0
↳ xcd6e9e19a712e0fe029cb241dfaa0746f64f197f3939ab069d0ac1ef843b4f36
  0
↳ xa59a7011e513c90dd9dff63c522bd947e7c6dc854da69f6005f2279803b3a2c3
  But we have:
  0
↳ xa59a7011e513c90dd9dff63c522bd947e7c6dc854da69f6005f2279803b3a2c3
  0
↳ x2859e3f8d44bc1720f80693c9242d107e6adcb1202343d1a14aac49edf9ffc9e
```

BVSS:

**AO:S/AC:L/AX:L/C:L/I:H/A:L/D:N/Y:N/R:N/S:U (1.8)**

Recommendation:

1. **Update `createNodeRequest`:** In the `createNodeRequest` function, after pushing `_nodeId` to `activeNodeRequestsIds`, set its index in the `activeNodeRequestsIdsIndex` mapping.

**Listing 25**

```
activeNodeRequestsIds.push(_nodeId);
activeNodeRequestsIdsIndex[_nodeId] = activeNodeRequestsIds.length
↳  - 1;
```

2. **Enhanced Check in `removeNodeRequest`:** Check if the `activeNodeRequestsIdsIndex` actually contains an index for `_nodeId`. If not, it means there's

an inconsistency and the function should revert or handle this situation appropriately.

**Listing 26**

```
require(activeNodeRequestsIdsIndex[_nodeId] != 0 ||
↳ activeNodeRequestsIds[0] == _nodeId, "Invalid NodeId provided");
```

This ensures that the function doesn't proceed if the provided `_nodeId` is invalid or not tracked correctly.

3. **Testing**: After making these changes, thorough testing should be done to ensure no other inconsistencies are introduced and the system behaves as expected.

By implementing these recommendations, the synchronization issue between the `activeNodeRequestsIds` array and `activeNodeRequestsIdsIndex` mapping can be resolved, ensuring consistent and reliable operations.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR32.

## 4.11 (HAL-11) MAXIMUM STAKING AMOUNT BELOW CURRENT STAKE - INFORMATIONAL (1.0)

Description:

The function setMaxStakingAmountForNode in MeldStakingConfig.sol is intended to override the maximum amount of MELD tokens that can be staked or delegated to a particular node. The current implementation only checks if the new maximum staking amount is higher than the global minimum staking amount. However, it fails to verify if the new maximum is greater than or equal to the current base staked amount on the node. This oversight allows an administrator to set a maximum staking amount that is less than the current staked amount, which would incorrectly categorize the node as being “overstaked” immediately after the update.

BVSS:

**AO:S/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (1.0)**

Recommendation:

It is crucial to add another check to ensure that _maxStakingAmount is not less than the current base staked amount for the node in question. This can be achieved by querying the current staked amount from the staking storage contract and comparing it to the proposed maximum staking amount.

By implementing this change, the setMaxStakingAmountForNode function will correctly enforce that the maximum staking amount cannot fall below the already staked amount, preventing the inadvertent classification of a node as overstaked.

Remediation Plan:

**NOT APPLICABLE**: This issue is an expected behavior and was acknowledged by the Meld team.

## 4.12 (HAL-12) INEFFICIENT GAS USAGE ON REMOVE TIER - INFORMATIONAL (0.5)

Description:

In the MeldStakingStorage contract, the removeStakingLockTier function is responsible for removing a specified staking lock tier from the active list of lock staking tiers. The function attempts to efficiently reindex the activeLockStakingTierIds array after removal, by swapping the element to be removed with the last element in the array and then popping the last element.

However, the function checks if lastLockStakingTierId is not zero, and if true, proceeds to perform the swap operation. This check is not representative of whether the swap operation is needed. This could lead to the swap operation being unnecessarily executed even if there's only one element in the activeLockStakingTierIds array, which would be inefficient in terms of gas.

The correct approach should be to check if there are more than one active staking lock tiers, which can be done by checking against lastLockStakingTierIdIndex.

BVSS:

**AO:S/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (0.5)**

Recommendation:

To fix this inefficiency, replace the check `if (lastLockStakingTierId != 0)` with `if (lastLockStakingTierIdIndex != 0)` to ensure the logic within the conditional is executed only when there's more than one element in the activeLockStakingTierIds array.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR33.

# 4.13 (HAL-13) IMPROPER INITIALIZATION CHECKS - INFORMATIONAL (1.5)

Description:

The MeldStakingStorage, MeldStakingNFTMetadata and MeldStakingCommon contracts defines an initialize function that sets up certain contract properties when called. It's expected that such initialization functions can be called only once (or under very controlled circumstances) to prevent inadvertent or malicious reconfiguration of the system. The checks within this function ensures that it can't be called more than once (using the initialized variable) and also verify that the provided _stakingAddressProvider address is non-zero.

However, the code doesn't check whether the addresses obtained from _stakingAddressProvider are valid (non-zero) before using them.

BVSS:

**AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (1.5)**

Recommendation:

1. Before using the addresses fetched from _stakingAddressProvider, verify that each address is non-zero.
2. Change the visibility of the initialized variable in MeldStakingAddressProvider from private to public. Then, add a check in the initialize function of the MeldStakingStorage contract:

**Listing 27**

```
require(!stakingAddressProvider.initialized(), "
↳ PROVIDER_ALREADY_INITIALIZED");
```

By making these changes, it helps ensure that the initialization process is more secure and robust against potential misconfiguration or malicious attempts.

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR34.

## 4.14 (HAL-14) INCORRECT DOCUMENTATION - INFORMATIONAL (0.0)

Description:

Documentation is crucial in Solidity code because it offers other developers, auditors, and users an understanding of how the contract should function. Inaccuracies in documentation can lead to misunderstandings which, in certain cases, can lead to misuse or errors.

In the MeldStakingConfig contract, the comment for the addStakingLockTier function's _stakingLength parameter suggests that the length is in seconds, when it is actually in terms of the number of epochs. Such discrepancies between the documentation and actual code can lead to confusion for users and developers.

BVSS:

**AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)**

Recommendation:

To prevent confusion, the documentation should be amended to accurately describe the _stakingLength parameter. The comment should be updated to:

**Listing 28**

```
1 * @param   _stakingLength  Duration of the lock period, in number
↳ of epochs
```

Remediation Plan:

**SOLVED**: The Meld team solved this issue in PR35.

# REVIEW NOTES

## 5.1 MeldStakingAddressProvider.sol

- Does allow setting several contract addresses that will be used across all contracts.
- **It does not allow updating any of the addresses in case a contract needs to be updated.**

## 5.2 MeldStakingOperator.sol

- The `initialize` function is not acting as a standard proxy initializer, but rather an admin only function to set the `addresProvider` contract.
- The `setTrustedForwarder`, which can only be called by a none-trusted forwarder and the role `TRUSTED_FORWARDER_SETTER_ROLE` to change the trusted forwarder.
- The `requestNode` function:
  - Does verify using the `onlyValidLockTier` that the initial amount is correct for that tire. A tier id of `0` is treated as a liquid staking and valid always.
  - The `stakingStorage.isNode` check is correct, as the none state can only be present when the node does not exist, no setter exists. Only the `createNode` does change that initial state.
  - The function does use the `stakingStorage.setNodeName` which sets the storage`nodeNames[_nodeId]`. **This means that the `getNodeName` could be used even after the node request is approved.**
- The `leaveNode` function:
  - The code does check for active and updates all previous metrics for the current node.
  - The `getStakerBaseStakedAmount` function will return delegators fees which will be added to the unclaimed amount to form the `totalAmount` withdrawn to the sender (checked to be the `isNftOwner`).

Issues:

- The initialize function assumes that the addresProvider is initialised and that the returned values will be non-zero. This means that if the addresProvider addresses are zero, as the contract was not initialised, it would be possible to keep calling initialize on this contract and set a new addresProvider. A better approach would be to verify that the addresProvider is not already set and revert the initialisation if the addresProvider values are zero.

## 5.3 MeldStakingCommon.sol

- **The updateUnclaimedRewards function is public, and can be called by anyone into any token id, not only the ones you own.**
  - It does make sure that the unclaimed rewards are not calculated after the node is slashed/inactive.
  - **It will only upgrade the NFT to liquid with _upgradeLockedToLiquid if the node is active. If not active, the delegation NFT (even if the locking time is reached) it will never be converted to liquid due to the _getLastActiveEpoch being used.**
- The _registerLockStaking is keeping track for none-liquid positions the total excess weighted stake that an epoch does have. On the updatePreviousEpochs the lockingExcessWeightedStakePerEpoch will be subtracted from the rolling amount.
- The updateStakerPreviousEpochs can be called by anyone, even not the owner of the NFT ID.

## 5.4 MeldStakingDelegator.sol

- stake does verify that the tier minimum amount is reached and that the staking has started. It also verifies that the node is “Active”, not slashed. It also checks that the stake amount plus the node baseStakedAmount does not go beyond the globalInfo.maxStakingAmount

or the specific amount set using setMaxStakingAmountForNode on the config.

- withdraw does verify the following:
  - The NFT id exists.
  - The caller of the function owns the NFT
  - The NFT type is delegation.
- withdraw does not verify if the node is active, which means it allows withdrawing after the node is deactivated.
- The changeDelegation does make sure that it is not delegated to the same node.

## 5.5 MeldStakingBase.sol

- The grantRole, revokeRole and renounceRole are checking that the sender is not a trusted meta transaction forwarder. Even though, those functions will only allow the action to happen if the sender is the admin, for the grant and revoke, or has the actual role for the renounce.
- The pause and unpause can only be called none-meta, and with the corresponding PAUSER_ROLE and UNPAUSER_ROLE.
- The _msgSender and _msgData are extracted from the ERC2771Recipient and any contract inheriting this will be using meta transaction aware functions.

## 5.6 RescueTokens.sol

Does implement three functions to transfer from the inheriting contract the following types:
- _rescueERC20: It is using the entire this balance. Probably some issues if the inheriting contract does allow tokens on its implementation, as admins could remove the usable balance from it.
- _rescueERC721: It is checking if the owner of the token is the contract itself. Some issues could exist if the ERC721 tokens are transferred to

the inheriting contract, as the admin could remove them from the contract. - _rescueERC1155: It is checking for the balance of this for the parameter tokenId. Some issues could exist if the tokens are transferred to the inheriting contract, as the admin could remove them from the contract.

## 5.7 MeldStakingStorage.sol

All setter and actions are limited to the onlyStakingOrConfig modifier, which will check for whitelisted contracts on a mapping. Only the contracts set on the initializer function can call those "action" functions.

- The createNodeRequest is limited to onlyStakingOrConfig

## 5.8 MeldStakingNFT.sol

- The setMetadataAddress can only be called by the admin and would allow changing the metadata contract always and without further restrictions.
- The rescueERC721 does restrict the token address to the address of self. Those preventing admins from ever removing or transferring unowned tokens.

## 5.9 MeldStakingNFTMetadata.sol

- If nodeName does contain json characters, it could be possible to perform JSON injection on the attributes.

## 5.10 MeldStakingConfig.sol

The slashNode function does not have a way to reset pending weighted values. However, since reward and new staking are prevented from the

moment the node was slashed, it does not cause any issue.

Issues:

- The setMaxStakingAmountForNode should not allow setting a value less than the base staked for that node. Otherwise, the node will be treated as “overstaked” instantly.

## 5.11 libraries/StakerLibrary.sol

- It does only expose one function to the library named updateMin. This function does update the minStakedAmountPerEpoch only if the lastStakedAmountPerEpoch is smaller. This means that the minStakedAmountPerEpoch will always track the min value of lastStakedAmountPerEpoch when called.

## 5.12 libraries/NodeLibrary.sol

- The updatePreviousEpochs function will verify if there is a new epoch, and return otherwise.
- The removeDelegator could potentially have an underflow if no delegators exist. However, all calls to the remove are proceeded with a getStakerNodeId for the NFT id and isNftOwner to verify that the NFT does exist. This means that self.delegators.length will always be different from zero when called and the _nftId existent.

# AUTOMATED TESTING

# 6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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, Slither was run on the all-scoped contracts. 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.

Slither results:

| Slither results for Meld - Staking | |
|---|---|
| **Finding** | **Impact** |
| MeldStakingNFT._depositMeld(address,uint256) (contracts/MeldStakingNFT.sol#298-305) uses arbitrary from in transferFrom: meldToken.safeTransferFrom(_from,address(this),_amount) (contracts/MeldStakingNFT.sol#303) | High |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division: - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division: - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |

| Finding | Impact |
| --- | --- |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |

| Finding | Impact |
| --- | --- |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Base64.encode(bytes) (node_modules/@openzeppelin/contracts/utils/Base64.sol#20-91) performs a multiplication on the result of a division:<br>- result = new string(4 * ((data.length + 2) / 3)) (node_modules/@openzeppelin/contracts/utils/Base64.sol#36) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |
| MeldStakingNFTMetadata._getRequestStakingParams(uint256).params (contracts/MeldStakingNFTMetadata.sol#203) is a local variable never initialized | Medium |
| MeldStakingNFTMetadata._getStakingParams(uint256).params (contracts/MeldStakingNFTMetadata.sol#161) is a local variable never initialized | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |

| Finding | Impact |
| --- | --- |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |
| MeldStakingStorage.isNodeSlashed(bytes32) (contracts/MeldStakingStorage.sol#490-492) uses a dangerous strict equality:<br>- nodes[_nodeId].status == NodeLibrary.NodeStatus.Slashed (contracts/MeldStakingStorage.sol#491) | Medium |
| MeldStakingStorage.isOperator(uint256) (contracts/MeldStakingStorage.sol#349-351) uses a dangerous strict equality:<br>- stakers[_nftId].stakerType == StakerLibrary.StakerType.Operator (contracts/MeldStakingStorage.sol#350) | Medium |
| MeldStakingStorage.isNodeActive(bytes32) (contracts/MeldStakingStorage.sol#472-474) uses a dangerous strict equality:<br>- nodes[_nodeId].status == NodeLibrary.NodeStatus.Active (contracts/MeldStakingStorage.sol#473) | Medium |
| MeldStakingStorage.isDelegator(uint256) (contracts/MeldStakingStorage.sol#358-360) uses a dangerous strict equality:<br>- stakers[_nftId].stakerType == StakerLibrary.StakerType.Delegator (contracts/MeldStakingStorage.sol#359) | Medium |
| MeldStakingStorage.isNodeInactive(bytes32) (contracts/MeldStakingStorage.sol#481-483) uses a dangerous strict equality:<br>- nodes[_nodeId].status == NodeLibrary.NodeStatus.Inactive (contracts/MeldStakingStorage.sol#482) | Medium |

| Finding | Impact |
|---|---|
| MeldStakingStorage.removeStaker(uint256) (contracts/MeldStakingStorage.sol#854-856) deletes StakerLibrary.Staker (contracts/libraries/StakerLibrary.sol#16-28) which contains a mapping: -delete stakers[_nftId] (contracts/MeldStakingStorage.sol#855) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division: - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#120) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division: - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)- result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#131) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division: - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division: - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division: - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124) | Medium |

| Finding | Impact |
|---|---|
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121) | Medium |
| Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication on the result of a division:<br>- denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)- inverse = (3 * denominator) extasciicircum 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#116) | Medium |
| MeldStakingCommon._updateStakerPreviousEpochs(uint256,uint256) (contracts/MeldStakingCommon.sol#484-506) has external calls inside a loop: lastEpochUpdated = stakingStorage.getStakerLastEpochStakingUpdated(_nftId) (contracts/MeldStakingCommon.sol#485) | Low |
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: stakingStorage.isNodeSlashed(nodeId) (contracts/MeldStakingCommon.sol#235) | Low |
| MeldStakingCommon.updateStakerPreviousEpochs(uint256) (contracts/MeldStakingCommon.sol#316-323) has external calls inside a loop: require(bool,string)(stakingStorage.isNode(nodeId),NODE_DOES_NOT_EXIST) (contracts/MeldStakingCommon.sol#320) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: lastEpochRewardsUpdated = stakingStorage.getLastEpochRewardsUpdated() (contracts/MeldStakingCommon.sol#283) | Low |

| Finding | Impact |
|---|---|
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: _getMeldStakingNFT().withdrawMeld(_msgSender(),unclaimedRewards) (contracts/MeldStakingCommon.sol#232) | Low |
| MeldStakingCommon._updateStakerPreviousEpochs(uint256,uint256) (contracts/MeldStakingCommon.sol#484-506) has external calls inside a loop: stakingStorage.setStakerLastEpochStakingUpdated(_nftId,_untilEpoch) (contracts/MeldStakingCommon.sol#505) | Low |
| MeldStakingCommon._upgradeLockedToLiquid(uint256,uint256) (contracts/MeldStakingCommon.sol#515-538) has external calls inside a loop: stakingStorage.updateNodePreviousEpochs(nodeId,_epoch) (contracts/MeldStakingCommon.sol#528) | Low |
| MeldStakingCommon._upgradeLockedToLiquid(uint256,uint256) (contracts/MeldStakingCommon.sol#515-538) has external calls inside a loop: stakingStorage.updateGlobalPreviousEpochs(_epoch) (contracts/MeldStakingCommon.sol#531) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: stakingStorage.updateGlobalPreviousEpochs(untilEpoch) (contracts/MeldStakingCommon.sol#279) | Low |
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: unclaimedRewards = stakingStorage.getStakerUnclaimedRewards(_nftId) (contracts/MeldStakingCommon.sol#225) | Low |
| MeldStakingCommon._getEndLockEpoch(uint256) (contracts/MeldStakingCommon.sol#612-621) has external calls inside a loop: startEpoch + stakingStorage.getLockStakingTier(lockTierId).stakingLength + 1 (contracts/MeldStakingCommon.sol#620) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: oldUnclaimedRewards = stakingStorage.getStakerUnclaimedRewards(_nftId) (contracts/MeldStakingCommon.sol#282) | Low |

| Finding | Impact |
| --- | --- |
| MeldStakingCommon._getEndLockEpoch(uint256) (contracts/MeldStakingCommon.sol#612-621) has external calls inside a loop: lockTierId = stakingStorage.getStakerLockTierId(_nftId) (contracts/MeldStakingCommon.sol#613) | Low |
| MeldStakingCommon._upgradeLockedToLiquid(uint256,uint256) (contracts/MeldStakingCommon.sol#515-538) has external calls inside a loop: nodeId = stakingStorage.getStakerNodeId(_nftId) (contracts/MeldStakingCommon.sol#527) | Low |
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: stakingStorage.removeDelegator(nodeId,_nftId) (contracts/MeldStakingCommon.sol#239) | Low |
| MeldStakingCommon._getEndLockEpoch(uint256) (contracts/MeldStakingCommon.sol#612-621) has external calls inside a loop: stakingStorage.getStakerLockTierId(_nftId) == 0 (contracts/MeldStakingCommon.sol#614) | Low |
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: stakingStorage.removeStaker(_nftId) (contracts/MeldStakingCommon.sol#242) | Low |
| MeldStakingCommon._updateStakerPreviousEpochs(uint256,uint256) (contracts/MeldStakingCommon.sol#484-506) has external calls inside a loop: stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#497) | Low |
| MeldStakingCommon._getEndLockEpoch(uint256) (contracts/MeldStakingCommon.sol#612-621) has external calls inside a loop: startEpoch = stakingStorage.getEpoch(stakingStorage.getStakerStakingStartTimestamp(_nftId)) (contracts/MeldStakingCommon.sol#617-619) | Low |
| MeldStakingCommon._upgradeLockedToLiquid(uint256,uint256) (contracts/MeldStakingCommon.sol#515-538) has external calls inside a loop: stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,_epoch,newStakerLastStakedAmount) (contracts/MeldStakingCommon.sol#524) | Low |
| MeldStakingCommon.ownerOfStakingNFT(uint256) (contracts/MeldStakingCommon.sol#430-432) has external calls inside a loop: _getMeldStakingNFT().ownerOf(_nftId) (contracts/MeldStakingCommon.sol#431) | Low |

| Finding | Impact |
|---|---|
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: nodeId = stakingStorage.getStakerNodeId(_nftId) (contracts/MeldStakingCommon.sol#222) | Low |
| MeldStakingCommon.updateStakerPreviousEpochs(uint256) (contracts/MeldStakingCommon.sol#316-323) has external calls inside a loop: nodeId = stakingStorage.getStakerNodeId(_nftId) (contracts/MeldStakingCommon.sol#319) | Low |
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: stakingStorage.isDelegator(_nftId) (contracts/MeldStakingCommon.sol#238) | Low |
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: stakingStorage.setStakerUnclaimedRewards(_nftId,0) (contracts/MeldStakingCommon.sol#231) | Low |
| MeldStakingCommon._getMeldStakingNFT() (contracts/MeldStakingCommon.sol#588-590) has external calls inside a loop: IMeldStakingNFT(stakingAddressProvider.meldStakingNFT()) (contracts/MeldStakingCommon.sol#589) | Low |
| MeldStakingCommon._getLastActiveEpoch(bytes32) (contracts/MeldStakingCommon.sol#597-605) has external calls inside a loop: stakingStorage.getCurrentEpoch() (contracts/MeldStakingCommon.sol#604) | Low |
| MeldStakingCommon.getWeightedAmount(uint256,uint256) (contracts/MeldStakingCommon.sol#464-475) has external calls inside a loop: (_amount * weight) / stakingStorage.PERCENTAGE_SCALING() (contracts/MeldStakingCommon.sol#474) | Low |
| MeldStakingCommon._getExcessWeightedStake(uint256) (contracts/MeldStakingCommon.sol#546-552) has external calls inside a loop: getWeightedAmount(baseStakedAmount,stakingStorage.getStakerLockTierId(_nftId)) - baseStakedAmount (contracts/MeldStakingCommon.sol#549-551) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: stakingStorage.setStakerUnclaimedRewards(_nftId,newUnclaimedRewards) (contracts/MeldStakingCommon.sol#301) | Low |

| Finding | Impact |
|---|---|
| MeldStakingCommon._updateStakerPreviousEpochs(uint256,uint256) (contracts/MeldStakingCommon.sol#484-506) has external calls inside a loop: rollingAmount = stakingStorage.getStakerLastStakedAmountPerEpoch(_nftId,lastEpochUpdated) (contracts/MeldStakingCommon.sol#489-492) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: require(bool,string)(stakingStorage.isNode(nodeId),NODE_DOES_NOT_EXIST) (contracts/MeldStakingCommon.sol#275) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: fromEpoch = stakingStorage.getStakerLastEpochRewardsUpdated(_nftId) + 1 (contracts/MeldStakingCommon.sol#281) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: rewards = (stakingStorage.getStakerMinStakedAmountPerEpoch(_nftId,epoch) * stakingStorage.getTotalRewardsPerEpoch(epoch)) / stakingStorage.getMinStakedAmountPerEpoch(epoch) (contracts/MeldStakingCommon.sol#295-297) | Low |
| MeldStakingCommon.stakingStarted() (contracts/MeldStakingCommon.sol#35-38) has external calls inside a loop: require(bool,string)(stakingStorage.isStakingStarted(),STAKING_NOT_STARTED) (contracts/MeldStakingCommon.sol#36) | Low |
| MeldStakingCommon._updateStakerPreviousEpochs(uint256,uint256) (contracts/MeldStakingCommon.sol#484-506) has external calls inside a loop: stakingStorage.setStakerMinStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#498) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: nodeId = stakingStorage.getStakerNodeId(_nftId) (contracts/MeldStakingCommon.sol#274) | Low |
| MeldStakingCommon._upgradeLockedToLiquid(uint256,uint256) (contracts/MeldStakingCommon.sol#515-538) has external calls inside a loop: newStakerLastStakedAmount = stakingStorage.getStakerLastStakedAmountPerEpoch(_nftId,_epoch) - excessWeightedStake (contracts/MeldStakingCommon.sol#520-523) | Low |

| Finding | Impact |
|---|---|
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: stakingStorage.updateNodePreviousEpochs(nodeId,untilEpoch) (contracts/MeldStakingCommon.sol#278) | Low |
| MeldStakingCommon.getWeightedAmount(uint256,uint256) (contracts/MeldStakingCommon.sol#464-475) has external calls inside a loop: weight = stakingStorage.getLockStakingTier(_lockTierId).weight (contracts/MeldStakingCommon.sol#473) | Low |
| MeldStakingCommon._getExcessWeightedStake(uint256) (contracts/MeldStakingCommon.sol#546-552) has external calls inside a loop: baseStakedAmount = stakingStorage.getStakerBaseStakedAmount(_nftId) (contracts/MeldStakingCommon.sol#547) | Low |
| MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310) has external calls inside a loop: stakingStorage.setStakerLastEpochRewardsUpdated(_nftId,calculateUntilEpoch) (contracts/MeldStakingCommon.sol#302) | Low |
| MeldStakingCommon._getLastActiveEpoch(bytes32) (contracts/MeldStakingCommon.sol#597-605) has external calls inside a loop: endTimestamp = stakingStorage.getNodeEndTimestamp(nodeId) (contracts/MeldStakingCommon.sol#598) | Low |
| MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246) has external calls inside a loop: _getMeldStakingNFT().redeem(_nftId) (contracts/MeldStakingCommon.sol#236) | Low |
| MeldStakingCommon.isStaker(uint256) (contracts/MeldStakingCommon.sol#53-56) has external calls inside a loop: require(bool,string)(stakingStorage.isStaker(_nftId),STAKER_DOES_NOT_EXIST) (contracts/MeldStakingCommon.sol#54) | Low |
| MeldStakingCommon._getLastActiveEpoch(bytes32) (contracts/MeldStakingCommon.sol#597-605) has external calls inside a loop: stakingStorage.getEpoch(endTimestamp) (contracts/MeldStakingCommon.sol#602) | Low |
| MeldStakingCommon._upgradeLockedToLiquid(uint256,uint256) (contracts/MeldStakingCommon.sol#515-538) has external calls inside a loop: stakingStorage.setStakerLockTierId(_nftId,0) (contracts/MeldStakingCommon.sol#534) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingCommon._registerLockStaking(uint256,uint256) (contracts/MeldStakingCommon.sol#560-582): External calls: - stakingStorage.setNodeLockingExcessWeightedStakePerEpoch(nodeId,endLockEpoch,stakingStorage.getNodeLockingExcessWeightedStakePerEpoch(nodeId,endLockEpoch) + excessWeightedStake) (contracts/MeldStakingCommon.sol#569-574) - stakingStorage.setLockingExcessWeightedStakePerEpoch(endLockEpoch,stakingStorage.getLockingExcessWeightedStakePerEpoch(endLockEpoch) + excessWeightedStake) (contracts/MeldStakingCommon.sol#577-580) Event emitted after the call(s): - LockStakingRegistered(_nftId,_lockTierId,endLockEpoch) (contracts/MeldStakingCommon.sol#581) | Low |
| Reentrancy in MeldStakingCommon._updateStakerPreviousEpochs(uint256,uint256) (contracts/MeldStakingCommon.sol#484-506): External calls: - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#497) - stakingStorage.setStakerMinStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#498) - newAmount = _upgradeLockedToLiquid(_nftId,epoch) (contracts/MeldStakingCommon.sol#500) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,_epoch,newStakerLastStakedAmount) (contracts/MeldStakingCommon.sol#524) - stakingStorage.updateNodePreviousEpochs(nodeId,_epoch) (contracts/MeldStakingCommon.sol#528) - stakingStorage.updateGlobalPreviousEpochs(_epoch) (contracts/MeldStakingCommon.sol#531) - stakingStorage.setStakerLockTierId(_nftId,0) (contracts/MeldStakingCommon.sol#534) Event emitted after the call(s): - StakerUpgradedToLiquid(_nftId,_epoch) (contracts/MeldStakingCommon.sol#536) - newAmount = _upgradeLockedToLiquid(_nftId,epoch) (contracts/MeldStakingCommon.sol#500) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingCommon.claimRewards(uint256) (contracts/MeldStakingCommon.sol#219-246): External calls:<br>- updateUnclaimedRewards(_nftId) (contracts/MeldStakingCommon.sol#224)<br>- stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,_epoch,newStakerLastStakedAmount) (contracts/MeldStakingCommon.sol#524)<br>- stakingStorage.updateNodePreviousEpochs(nodeId,untilEpoch) (contracts/MeldStakingCommon.sol#278)<br>- stakingStorage.updateNodePreviousEpochs(nodeId,_epoch) (contracts/MeldStakingCommon.sol#528)<br>- stakingStorage.updateGlobalPreviousEpochs(_epoch) (contracts/MeldStakingCommon.sol#531)<br>- stakingStorage.updateGlobalPreviousEpochs(untilEpoch) (contracts/MeldStakingCommon.sol#279)<br>- stakingStorage.setStakerLockTierId(_nftId,0) (contracts/MeldStakingCommon.sol#534)<br>- stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#497)<br>- stakingStorage.setStakerMinStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#498)<br>- stakingStorage.setStakerLastEpochStakingUpdated(_nftId,_untilEpoch) (contracts/MeldStakingCommon.sol#505)<br>- stakingStorage.setStakerUnclaimedRewards(_nftId,newUnclaimedRewards) (contracts/MeldStakingCommon.sol#301)<br>- stakingStorage.setStakerLastEpochRewardsUpdated(_nftId,calculateUntilEpoch) (contracts/MeldStakingCommon.sol#302)<br>- stakingStorage.setStakerUnclaimedRewards(_nftId,0) (contracts/MeldStakingCommon.sol#231)<br>- _getMeldStakingNFT().withdrawMeld(_msgSender(),unclaimedRewards) (contracts/MeldStakingCommon.sol#232)<br>- _getMeldStakingNFT().redeem(_nftId) (contracts/MeldStakingCommon.sol#236)<br>- stakingStorage.removeDelegator(nodeId,_nftId) (contracts/MeldStakingCommon.sol#239)<br>- stakingStorage.removeStaker(_nftId) (contracts/MeldStakingCommon.sol#242) Event emitted after the call(s):<br>- RewardsClaimed(_nftId,unclaimedRewards) (contracts/MeldStakingCommon.sol#245) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingCommon.newStake(uint256,uint256) (contracts/MeldStakingCommon.sol#158-210): External calls: - stakingStorage.updateNodePreviousEpochs(nodeId,currentEpoch) (contracts/MeldStakingCommon.sol#165) - stakingStorage.updateGlobalPreviousEpochs(currentEpoch) (contracts/MeldStakingCommon.sol#166) - _updateStakerPreviousEpochs(operator,currentEpoch) (contracts/MeldStakingCommon.sol#174) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,_epoch,newStakerLastStakedAmount) (contracts/MeldStakingCommon.sol#524) - stakingStorage.updateNodePreviousEpochs(nodeId,_epoch) (contracts/MeldStakingCommon.sol#528) - stakingStorage.updateGlobalPreviousEpochs(_epoch) (contracts/MeldStakingCommon.sol#531) - stakingStorage.setStakerLockTierId(_nftId,0) (contracts/MeldStakingCommon.sol#534) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#497) - stakingStorage.setStakerMinStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#498) - stakingStorage.setStakerLastEpochStakingUpdated(_nftId,_untilEpoch) (contracts/MeldStakingCommon.sol#505) - stakingStorage.setStakerLastStakedAmountPerEpoch(operator,currentEpoch,stakingStorage.getStakerLastStakedAmountPerEpoch(operator,currentEpoch) + feeAmount) (contracts/MeldStakingCommon.sol#176-180) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,currentEpoch,weightedAmount - feeAmount) (contracts/MeldStakingCommon.sol#183-187) - stakingStorage.setNodeLastStakedAmountPerEpoch(nodeId,currentEpoch,stakingStorage.getNodeLastStakedAmountPerEpoch(nodeId,currentEpoch) + weightedAmount) (contracts/MeldStakingCommon.sol#188-192) - stakingStorage.setLastStakedAmountPerEpoch(currentEpoch,stakingStorage.getLastStakedAmountPerEpoch(currentEpoch) + weightedAmount) (contracts/MeldStakingCommon.sol#193-196) - stakingStorage.setStakerBaseStakedAmount(_nftId,_newAmount) (contracts/MeldStakingCommon.sol#198) - stakingStorage.setNodeBaseStakedAmount(nodeId,stakingStorage.getNodeBaseStakedAmount(nodeId) + _newAmount) (contracts/MeldStakingCommon.sol#199-202) - stakingStorage.setTotalBaseStakedAmount(stakingStorage.getTotalBaseStakedAmount() + _newAmount) (contracts/MeldStakingCommon.sol#203-205) - _registerLockStaking(_nftId,lockTierId) (contracts/MeldStakingCommon.sol#208) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingCommon.updateUnclaimedRewards(uint256) (contracts/MeldStakingCommon.sol#273-310): External calls: - _updateStakerPreviousEpochs(_nftId,untilEpoch) (contracts/MeldStakingCommon.sol#277) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,_epoch,newStakerLastStakedAmount) (contracts/MeldStakingCommon.sol#524) - stakingStorage.updateNodePreviousEpochs(nodeId,_epoch) (contracts/MeldStakingCommon.sol#528) - stakingStorage.updateGlobalPreviousEpochs(_epoch) (contracts/MeldStakingCommon.sol#531) - stakingStorage.setStakerLockTierId(_nftId,0) (contracts/MeldStakingCommon.sol#534) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#497) - stakingStorage.setStakerMinStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#498) - stakingStorage.setStakerLastEpochStakingUpdated(_nftId,_untilEpoch) (contracts/MeldStakingCommon.sol#505) - stakingStorage.updateNodePreviousEpochs(nodeId,untilEpoch) (contracts/MeldStakingCommon.sol#278) - stakingStorage.updateGlobalPreviousEpochs(untilEpoch) (contracts/MeldStakingCommon.sol#279) - stakingStorage.setStakerUnclaimedRewards(_nftId,newUnclaimedRewards) (contracts/MeldStakingCommon.sol#301) - stakingStorage.setStakerLastEpochRewardsUpdated(_nftId,calculateUntilEpoch) (contracts/MeldStakingCommon.sol#302) Event emitted after the call(s): - UnclaimedRewardsUpdated(_nftId,oldUnclaimedRewards,newUnclaimedRewards,fromEpoch,calculateUntilEpoch) (contracts/MeldStakingCommon.sol#303-309) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingCommon.newStake(uint256,uint256) (contracts/MeldStakingCommon.sol#158-210): External calls: - stakingStorage.updateNodePreviousEpochs(nodeId,currentEpoch) (contracts/MeldStakingCommon.sol#165) - stakingStorage.updateGlobalPreviousEpochs(currentEpoch) (contracts/MeldStakingCommon.sol#166) - _updateStakerPreviousEpochs(operator,currentEpoch) (contracts/MeldStakingCommon.sol#174) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,_epoch,newStakerLastStakedAmount) (contracts/MeldStakingCommon.sol#524) - stakingStorage.updateNodePreviousEpochs(nodeId,_epoch) (contracts/MeldStakingCommon.sol#528) - stakingStorage.updateGlobalPreviousEpochs(_epoch) (contracts/MeldStakingCommon.sol#531) - stakingStorage.setStakerLockTierId(_nftId,0) (contracts/MeldStakingCommon.sol#534) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#497) - stakingStorage.setStakerMinStakedAmountPerEpoch(_nftId,epoch,rollingAmount) (contracts/MeldStakingCommon.sol#498) - stakingStorage.setStakerLastEpochStakingUpdated(_nftId,_untilEpoch) (contracts/MeldStakingCommon.sol#505) Event emitted after the call(s): - StakerUpgradedToLiquid(_nftId,_epoch) (contracts/MeldStakingCommon.sol#536) - _updateStakerPreviousEpochs(operator,currentEpoch) (contracts/MeldStakingCommon.sol#174) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingCommon._upgradeLockedToLiquid(uint256,uint256) (contracts/MeldStakingCommon.sol#515-538): External calls: - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,_epoch,newStakerLastStakedAmount) (contracts/MeldStakingCommon.sol#524) - stakingStorage.updateNodePreviousEpochs(nodeId,_epoch) (contracts/MeldStakingCommon.sol#528) - stakingStorage.updateGlobalPreviousEpochs(_epoch) (contracts/MeldStakingCommon.sol#531) - stakingStorage.setStakerLockTierId(_nftId,0) (contracts/MeldStakingCommon.sol#534) Event emitted after the call(s): - StakerUpgradedToLiquid(_nftId,_epoch) (contracts/MeldStakingCommon.sol#536) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingDelegator.withdraw(uint256) (contracts/MeldStakingDelegator.sol#270-342): External calls: - stakingCommon.updateStakerPreviousEpochs(_nftId,currentEpoch) (contracts/MeldStakingDelegator.sol#284) - stakingCommon.updateStakerPreviousEpochs(operator,currentEpoch) (contracts/MeldStakingDelegator.sol#285) - stakingStorage.updateNodePreviousEpochs(nodeId,currentEpoch) (contracts/MeldStakingDelegator.sol#286) - stakingStorage.updateGlobalPreviousEpochs(currentEpoch) (contracts/MeldStakingDelegator.sol#287) - stakingCommon.updateUnclaimedRewards(_nftId) (contracts/MeldStakingDelegator.sol#290) - stakingStorage.setStakerLastStakedAmountPerEpoch(operator,currentEpoch,stakingStorage.getStakerLastStakedAmountPerEpoch(operator,currentEpoch) - feeAmount) (contracts/MeldStakingDelegator.sol#299-303) - stakingStorage.setNodeBaseStakedAmount(nodeId,stakingStorage.getNodeBaseStakedAmount(nodeId) - baseStakedAmount) (contracts/MeldStakingDelegator.sol#306-309) - stakingStorage.setNodeLastStakedAmountPerEpoch(nodeId,currentEpoch,stakingStorage.getNodeLastStakedAmountPerEpoch(nodeId,currentEpoch) - baseStakedAmount) (contracts/MeldStakingDelegator.sol#310-314) - stakingStorage.setTotalBaseStakedAmount(oldTotalBaseStakedAmount - baseStakedAmount) (contracts/MeldStakingDelegator.sol#317) - stakingStorage.setLastStakedAmountPerEpoch(currentEpoch,stakingStorage.getLastStakedAmountPerEpoch(currentEpoch) - baseStakedAmount) (contracts/MeldStakingDelegator.sol#318-321) - stakingStorage.removeStaker(_nftId) (contracts/MeldStakingDelegator.sol#324) - stakingStorage.removeDelegator(nodeId,_nftId) (contracts/MeldStakingDelegator.sol#325) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,currentEpoch,0) (contracts/MeldStakingDelegator.sol#326) - stakingCommon.redeemStakingNFT(_nftId) (contracts/MeldStakingDelegator.sol#329) - stakingCommon.withdrawMeld(_msgSender(),totalAmount) (contracts/MeldStakingDelegator.sol#330) Event emitted after the call(s): - RewardsClaimed(_nftId,unclaimedRewards) (contracts/MeldStakingDelegator.sol#334) - StakeWithdrawn(_nftId,nodeId,totalAmount) (contracts/MeldStakingDelegator.sol#336) - TotalBaseStakedAmountChanged(_msgSender(),oldTotalBaseStakedAmount,stakingStorage.getTotalBaseStakedAmount()) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingDelegator.changeDelegation(uint256,bytes32) (contracts/MeldStakingDelegator.sol#174-261): External calls: - stakingCommon.updateStakerPreviousEpochs(_nftId,currentEpoch) (contracts/MeldStakingDelegator.sol#189) - stakingStorage.updateNodePreviousEpochs(_newNodeId,currentEpoch) (contracts/MeldStakingDelegator.sol#190) - stakingStorage.updateNodePreviousEpochs(oldNodeId,currentEpoch) (contracts/MeldStakingDelegator.sol#191) - stakingStorage.updateGlobalPreviousEpochs(currentEpoch) (contracts/MeldStakingDelegator.sol#192) - stakingCommon.updateUnclaimedRewards(_nftId) (contracts/MeldStakingDelegator.sol#195) - stakingStorage.setStakerLastStakedAmountPerEpoch(_nftId,currentEpoch,t.stakerBaseStakedAmount + t.oldFeeAmount - t.newFeeAmount) (contracts/MeldStakingDelegator.sol#201-205) - stakingStorage.setStakerLastStakedAmountPerEpoch(oldOperator,currentEpoch,stakingStorage.getStakerLastStakedAmountPerEpoch(oldOperator,currentEpoch) - t.oldFeeAmount) (contracts/MeldStakingDelegator.sol#210-215) - stakingStorage.setStakerLastStakedAmountPerEpoch(newOperator,currentEpoch,stakingStorage.getStakerLastStakedAmountPerEpoch(newOperator,currentEpoch) + t.newFeeAmount) (contracts/MeldStakingDelegator.sol#220-225) - stakingStorage.setNodeBaseStakedAmount(oldNodeId,stakingStorage.getNodeBaseStakedAmount(oldNodeId) - t.stakerBaseStakedAmount) (contracts/MeldStakingDelegator.sol#230-233) - stakingStorage.setNodeLastStakedAmountPerEpoch(oldNodeId,currentEpoch,stakingStorage.getNodeLastStakedAmountPerEpoch(oldNodeId,currentEpoch) - t.weightedAmountWithoutFee) (contracts/MeldStakingDelegator.sol#234-239) - stakingStorage.setNodeBaseStakedAmount(_newNodeId,stakingStorage.getNodeBaseStakedAmount(_newNodeId) + t.stakerBaseStakedAmount) (contracts/MeldStakingDelegator.sol#242-245) - stakingStorage.setNodeLastStakedAmountPerEpoch(_newNodeId,currentEpoch,stakingStorage.getNodeLastStakedAmountPerEpoch(_newNodeId,currentEpoch) + t.weightedAmountWithoutFee) (contracts/MeldStakingDelegator.sol#246-251) - stakingStorage.removeDelegator(oldNodeId,_nftId) (contracts/MeldStakingDelegator.sol#256) - stakingStorage.addDelegator(_newNodeId,_nftId) (contracts/MeldStakingDelegator.sol#257) - stakingStorage.setStakerNodeId(_nftId,_newNodeId) (contracts/MeldStakingDelegator.sol#259) Event emitted after the | Low |

| Finding | Impact |
|---|---|
| Reentrancy in MeldStakingDelegator.stake(uint256,bytes32,uint256) (contracts/MeldStakingDelegator.sol#133-166): External calls: - nftId = stakingCommon.mintStakingNFT(_msgSender(),_amount) (contracts/MeldStakingDelegator.sol#145) - stakingStorage.createStaker(nftId,2,_nodeId,_lockTierId) (contracts/MeldStakingDelegator.sol#147-152) - stakingCommon.newStake(nftId,_amount) (contracts/MeldStakingDelegator.sol#156) - stakingStorage.addDelegator(_nodeId,nftId) (contracts/MeldStakingDelegator.sol#158) Event emitted after the call(s): - StakingDelegationCreated(_msgSender(),nftId,_nodeId,_amount,_lockTierId) (contracts/MeldStakingDelegator.sol#160) - TotalBaseStakedAmountChanged(_msgSender(),oldTotalBaseStakedAmount,stakingStorage.getTotalBaseStakedAmount()) (contracts/MeldStakingDelegator.sol#161-165) | Low |
| MeldStakingDelegator.withdraw(uint256) (contracts/MeldStakingDelegator.sol#270-342) uses timestamp for comparisons Dangerous comparisons: - require(bool,string)(block.timestamp >= endOfLocking,STAKING_LOCKED) (contracts/MeldStakingDelegator.sol#276) | Low |
| Reentrancy in MeldStakingNFT._depositMeld(address,uint256) (contracts/MeldStakingNFT.sol#298-305): External calls: - meldToken.safeTransferFrom(_from,address(this),_amount) (contracts/MeldStakingNFT.sol#303) Event emitted after the call(s): - MeldDeposited(_from,_amount) (contracts/MeldStakingNFT.sol#304) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingNFT.mint(address,uint256) (contracts/MeldStakingNFT.sol#138-147): External calls: - _depositMeld(_to,_amount) (contracts/MeldStakingNFT.sol#142) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#122)- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)- meldToken.safeTransferFrom(_from,address(this),_amount) (contracts/MeldStakingNFT.sol#303) - _safeMint(_to,tokenId) (contracts/MeldStakingNFT.sol#145) - retval = IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#406-417)External calls sending eth: - _depositMeld(_to,_amount) (contracts/MeldStakingNFT.sol#142) - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)Event emitted after the call(s): - Transfer(address(0),to,tokenId) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#283)- _safeMint(_to,tokenId) (contracts/MeldStakingNFT.sol#145) | Low |
| Reentrancy in MeldStakingNFT.rescueMeldTokens(address) (contracts/MeldStakingNFT.sol#191-195): External calls: - meldToken.safeTransfer(_to,amount) (contracts/MeldStakingNFT.sol#193) Event emitted after the call(s): - MeldRescued(_to,amount) (contracts/MeldStakingNFT.sol#194) | Low |
| Reentrancy in MeldStakingNFT.withdrawMeld(address,uint256) (contracts/MeldStakingNFT.sol#177-182): External calls: - meldToken.safeTransfer(_to,_amount) (contracts/MeldStakingNFT.sol#180) Event emitted after the call(s): - MeldWithdrawn(_to,_amount) (contracts/MeldStakingNFT.sol#181) | Low |
| MeldStakingConfig._removeDelegatorFromWhitelist(bytes32,address) (contracts/MeldStakingConfig.sol#458-461) has external calls inside a loop: stakingStorage.removeDelegatorFromWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#459) | Low |

| Finding | Impact |
| --- | --- |
| MeldStakingConfig._addDelegatorToWhitelist(bytes32,address) (contracts/MeldStakingConfig.sol#447-450) has external calls inside a loop: stakingStorage.addDelegatorToWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#448) | Low |
| Reentrancy in MeldStakingConfig.setMinDelegationFee(uint256) (contracts/MeldStakingConfig.sol#146-154): External calls: - stakingStorage.setMinDelegationFee(_minDelegationFee) (contracts/MeldStakingConfig.sol#152) Event emitted after the call(s): - MinDelegationFeeUpdated(_msgSender(),minDelegationFee,_minDelegationFee) (contracts/MeldStakingConfig.sol#153) | Low |
| Reentrancy in MeldStakingConfig.setMaxStakingAmount(uint256) (contracts/MeldStakingConfig.sol#109-117): External calls: - stakingStorage.setMaxStakingAmount(_maxStakingAmount) (contracts/MeldStakingConfig.sol#115) Event emitted after the call(s): - MaxStakingAmountUpdated(_msgSender(),maxStakingAmount,_maxStakingAmount) (contracts/MeldStakingConfig.sol#116) | Low |
| Reentrancy in MeldStakingConfig._removeDelegatorFromWhitelist(bytes32,address) (contracts/MeldStakingConfig.sol#458-461): External calls: - stakingStorage.removeDelegatorFromWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#459) Event emitted after the call(s): - NodeDelegatorRemovedFromWhitelist(_msgSender(),_nodeId,_address) (contracts/MeldStakingConfig.sol#460) | Low |
| Reentrancy in MeldStakingConfig.setSlashReceiver(address) (contracts/MeldStakingConfig.sol#176-183): External calls: - stakingStorage.setSlashReceiver(_slashReceiver) (contracts/MeldStakingConfig.sol#181) Event emitted after the call(s): - SlashReceiverUpdated(_msgSender(),slashReceiver,_slashReceiver) (contracts/MeldStakingConfig.sol#182) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingConfig.slashNode(bytes32) (contracts/MeldStakingConfig.sol#305-345): External calls: - stakingStorage.updateNodePreviousEpochs(_nodeId,currentEpoch) (contracts/MeldStakingConfig.sol#311) - stakingStorage.updateGlobalPreviousEpochs(currentEpoch) (contracts/MeldStakingConfig.sol#312) - stakingStorage.setNodeBaseStakedAmount(_nodeId,0) (contracts/MeldStakingConfig.sol#323) - stakingStorage.setNodeLastStakedAmountPerEpoch(_nodeId,currentEpoch,0) (contracts/MeldStakingConfig.sol#324) - stakingStorage.setNodeSlashed(_nodeId) (contracts/MeldStakingConfig.sol#325) - stakingStorage.setTotalBaseStakedAmount(newTotalBaseStakedAmount) (contracts/MeldStakingConfig.sol#329) - stakingStorage.setLastStakedAmountPerEpoch(currentEpoch,stakingStorage.getLastStakedAmountPerEpoch(currentEpoch) - nodeLastStakedAmount) (contracts/MeldStakingConfig.sol#330-333) - stakingCommon.withdrawMeld(stakingStorage.slashReceiver(),nodeBaseStakedAmount) (contracts/MeldStakingConfig.sol#336) Event emitted after the call(s): - NodeSlashed(_msgSender(),_nodeId,nodeBaseStakedAmount) (contracts/MeldStakingConfig.sol#339) - TotalBaseStakedAmountChanged(_msgSender(),oldTotalBaseStakedAmount,newTotalBaseStakedAmount) (contracts/MeldStakingConfig.sol#340-344) | Low |
| Reentrancy in MeldStakingConfig.rejectNodeRequest(bytes32) (contracts/MeldStakingConfig.sol#280-297): External calls: - stakingStorage.removeNodeRequest(_nodeId) (contracts/MeldStakingConfig.sol#285) - stakingCommon.redeemStakingNFT(nodeRequest.operator) (contracts/MeldStakingConfig.sol#288) - stakingCommon.withdrawMeld(operatorOwner,nodeRequest.stakingAmount) (contracts/MeldStakingConfig.sol#289) Event emitted after the call(s): - NodeRequestRejected(_msgSender(),_nodeId,nodeRequest.operator,nodeRequest.stakingAmount) (contracts/MeldStakingConfig.sol#291-296) | Low |

| Finding | Impact |
|---|---|
| Reentrancy in MeldStakingConfig.toggleDelegatorWhitelist(bytes32,bool) (contracts/MeldStakingConfig.sol#353-359): External calls: - stakingStorage.toggleDelegatorWhitelist(_nodeId,_flag) (contracts/MeldStakingConfig.sol#357) Event emitted after the call(s): - NodeDelegatorWhitlistToggled(_msgSender(),_nodeId,_flag) (contracts/MeldStakingConfig.sol#358) | Low |
| Reentrancy in MeldStakingConfig.setMinStakingAmount(uint256) (contracts/MeldStakingConfig.sol#94-102): External calls: - stakingStorage.setMinStakingAmount(_minStakingAmount) (contracts/MeldStakingConfig.sol#100) Event emitted after the call(s): - MinStakingAmountUpdated(_msgSender(),minStakingAmount,_minStaking Amount) (contracts/MeldStakingConfig.sol#101) | Low |
| Reentrancy in MeldStakingConfig.setRewards(uint256,uint256) (contracts/MeldStakingConfig.sol#426-437): External calls: - stakingStorage.setRewards(_epoch,_amount) (contracts/MeldStakingConfig.sol#433) - stakingCommon.depositMeld(_msgSender(),_amount) (contracts/MeldStakingConfig.sol#434) Event emitted after the call(s): - RewardsSet(_msgSender(),_epoch,_amount) (contracts/MeldStakingConfig.sol#436) | Low |
| Reentrancy in MeldStakingConfig.addDelegatorToWhitelist(bytes32,address) (contracts/MeldStakingConfig.sol#367-373): External calls: - stakingStorage.enableNodeWhitelistIfNeeded(_nodeId) (contracts/MeldStakingConfig.sol#371) - _addDelegatorToWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#372) - stakingStorage.addDelegatorToWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#448) Event emitted after the call(s): - NodeDelegatorAddedToWhitelist(_msgSender(),_nodeId,_address) (contracts/MeldStakingConfig.sol#449) - _addDelegatorToWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#372) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingConfig.addDelegatorsToWhitelist(bytes32,address[]) (contracts/MeldStakingConfig.sol#381-389): External calls: - stakingStorage.enableNodeWhitelistIfNeeded(_nodeId) (contracts/MeldStakingConfig.sol#385) - _addDelegatorToWhitelist(_nodeId,_addresses[i]) (contracts/MeldStakingConfig.sol#387) - stakingStorage.addDelegatorToWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#448) Event emitted after the call(s): - NodeDelegatorAddedToWhitelist(_msgSender(),_nodeId,_address) (contracts/MeldStakingConfig.sol#449) - _addDelegatorToWhitelist(_nodeId,_addresses[i]) (contracts/MeldStakingConfig.sol#387) | Low |
| Reentrancy in MeldStakingConfig._addDelegatorToWhitelist(bytes32,address) (contracts/MeldStakingConfig.sol#447-450): External calls: - stakingStorage.addDelegatorToWhitelist(_nodeId,_address) (contracts/MeldStakingConfig.sol#448) Event emitted after the call(s): - NodeDelegatorAddedToWhitelist(_msgSender(),_nodeId,_address) (contracts/MeldStakingConfig.sol#449) | Low |
| Reentrancy in MeldStakingConfig.setMaxDelegationFee(uint256) (contracts/MeldStakingConfig.sol#161-170): External calls: - stakingStorage.setMaxDelegationFee(_maxDelegationFee) (contracts/MeldStakingConfig.sol#168) Event emitted after the call(s): - MaxDelegationFeeUpdated(_msgSender(),maxDelegationFee,_maxDelegationFee) (contracts/MeldStakingConfig.sol#169) | Low |
| Reentrancy in MeldStakingConfig.removeStakingLockTier(uint256) (contracts/MeldStakingConfig.sol#224-236): External calls: - stakingStorage.removeStakingLockTier(_lockTierId) (contracts/MeldStakingConfig.sol#233) Event emitted after the call(s): - StakingLockTierRemoved(_msgSender(),_lockTierId) (contracts/MeldStakingConfig.sol#235) | Low |

| Finding | Impact |
|---|---|
| Reentrancy in MeldStakingConfig.setMaxStakingAmountForNode(bytes32,uint256) (contracts/MeldStakingConfig.sol#125-139): External calls:<br>- stakingStorage.setNodeMaxStakingAmount(_nodeId,_maxStakingAmount) (contracts/MeldStakingConfig.sol#132) Event emitted after the call(s):<br>- MaxStakingAmountForNodeUpdated(_msgSender(),_nodeId,oldMaxStakingAmount,_maxStakingAmount) (contracts/MeldStakingConfig.sol#133-138) | Low |
| Reentrancy in MeldStakingConfig.addStakingLockTier(uint256,uint256,uint256) (contracts/MeldStakingConfig.sol#191-217): External calls:<br>- lastLockStakingTierId = stakingStorage.addLockStakingTier(_minStakingAmount,_stakingLength,_weight) (contracts/MeldStakingConfig.sol#204-208) Event emitted after the call(s):<br>- StakingLockTierAdded(_msgSender(),lastLockStakingTierId,_minStakingAmount,_stakingLength,_weight) (contracts/MeldStakingConfig.sol#210-216) | Low |
| Reentrancy in MeldStakingConfig.approveNodeRequest(bytes32) (contracts/MeldStakingConfig.sol#243-273): External calls:<br>- stakingStorage.createNode(_nodeId,operator,nodeRequest.delegatorFee) (contracts/MeldStakingConfig.sol#252)<br>- stakingStorage.createStaker(operator,1,_nodeId,nodeRequest.lockTierId) (contracts/MeldStakingConfig.sol#254-259)<br>- stakingCommon.newStake(operator,stakingAmount) (contracts/MeldStakingConfig.sol#263)<br>- stakingStorage.removeNodeRequest(_nodeId) (contracts/MeldStakingConfig.sol#264) Event emitted after the call(s):<br>- NodeRequestApproved(_msgSender(),_nodeId,operator,stakingAmount) (contracts/MeldStakingConfig.sol#266)<br>- TotalBaseStakedAmountChanged(_msgSender(),oldTotalBaseStakedAmount,stakingStorage.getTotalBaseStakedAmount()) (contracts/MeldStakingConfig.sol#268-272) | Low |

| Finding | Impact |
|---|---|
| Reentrancy in MeldStakingConfig.initialize(uint256,uint256,address,address) (contracts/MeldStakingConfig.sol#53-87): External calls: - stakingStorage.initializeConfig(_initTimestamp,_epochSize,_slashReceiver) (contracts/MeldStakingConfig.sol#78) Event emitted after the call(s): - Initialized(_msgSender(),_initTimestamp,_epochSize,_slashReceiver,_stakingAddressProvider) (contracts/MeldStakingConfig.sol#80-86) | Low |
| MeldStakingConfig.initialize(uint256,uint256,address,address) (contracts/MeldStakingConfig.sol#53-87) uses timestamp for comparisons Dangerous comparisons: - require(bool,string)(_initTimestamp > block.timestamp,INVALID_INIT_TIMESTAMP) (contracts/MeldStakingConfig.sol#59) | Low |
| MeldStakingStorage.setSlashReceiver(address)._slashReceiver (contracts/MeldStakingStorage.sol#672) lacks a zero-check on : - slashReceiver = _slashReceiver (contracts/MeldStakingStorage.sol#673) | Low |
| MeldStakingStorage.initializeConfig(uint256,uint256,address)._slashReceiver (contracts/MeldStakingStorage.sol#107) lacks a zero-check on : - slashReceiver = _slashReceiver (contracts/MeldStakingStorage.sol#116) | Low |
| MeldStakingStorage.isNodeSlashed(bytes32) (contracts/MeldStakingStorage.sol#490-492) uses timestamp for comparisons Dangerous comparisons: - nodes[_nodeId].status == NodeLibrary.NodeStatus.Slashed (contracts/MeldStakingStorage.sol#491) | Low |
| MeldStakingStorage.getEpoch(uint256) (contracts/MeldStakingStorage.sol#271-276) uses timestamp for comparisons Dangerous comparisons: - globalInfo.initTimestamp == 0 \|\| _timestamp < globalInfo.initTimestamp (contracts/MeldStakingStorage.sol#272) | Low |
| MeldStakingStorage.isStaker(uint256) (contracts/MeldStakingStorage.sol#340-342) uses timestamp for comparisons Dangerous comparisons: - stakers[_nftId].stakerType != StakerLibrary.StakerType.None (contracts/MeldStakingStorage.sol#341) | Low |

| Finding | Impact |
| --- | --- |
| MeldStakingStorage.isNodeActive(bytes32) (contracts/MeldStakingStorage.sol#472-474) uses timestamp for comparisons Dangerous comparisons: - nodes[_nodeId].status == NodeLibrary.NodeStatus.Active (contracts/MeldStakingStorage.sol#473) | Low |
| MeldStakingStorage.nodeRequestExists(bytes32) (contracts/MeldStakingStorage.sol#652-654) uses timestamp for comparisons Dangerous comparisons: - nodeRequests[_nodeId].requestTimestamp != 0 (contracts/MeldStakingStorage.sol#653) | Low |
| MeldStakingStorage.isNode(bytes32) (contracts/MeldStakingStorage.sol#463-465) uses timestamp for comparisons Dangerous comparisons: - nodes[_nodeId].status != NodeLibrary.NodeStatus.None (contracts/MeldStakingStorage.sol#464) | Low |
| MeldStakingStorage.isNodeInactive(bytes32) (contracts/MeldStakingStorage.sol#481-483) uses timestamp for comparisons Dangerous comparisons: - nodes[_nodeId].status == NodeLibrary.NodeStatus.Inactive (contracts/MeldStakingStorage.sol#482) | Low |
| MeldStakingStorage.isDelegator(uint256) (contracts/MeldStakingStorage.sol#358-360) uses timestamp for comparisons Dangerous comparisons: - stakers[_nftId].stakerType == StakerLibrary.StakerType.Delegator (contracts/MeldStakingStorage.sol#359) | Low |
| MeldStakingStorage.isStakingStarted() (contracts/MeldStakingStorage.sol#252-254) uses timestamp for comparisons Dangerous comparisons: - globalInfo.initTimestamp != 0 && block.timestamp >= globalInfo.initTimestamp (contracts/MeldStakingStorage.sol#253) | Low |
| MeldStakingStorage.isOperator(uint256) (contracts/MeldStakingStorage.sol#349-351) uses timestamp for comparisons Dangerous comparisons: - stakers[_nftId].stakerType == StakerLibrary.StakerType.Operator (contracts/MeldStakingStorage.sol#350) | Low |

| Finding | Impact |
| --- | --- |
| Reentrancy in MeldStakingOperator.leaveNode(uint256) (contracts/MeldStakingOperator.sol#190-261): External calls: - stakingCommon.updateStakerPreviousEpochs(_nftId,currentEpoch) (contracts/MeldStakingOperator.sol#203) - stakingStorage.updateNodePreviousEpochs(nodeId,currentEpoch) (contracts/MeldStakingOperator.sol#204) - stakingStorage.updateGlobalPreviousEpochs(currentEpoch) (contracts/MeldStakingOperator.sol#205) - stakingCommon.updateUnclaimedRewards(_nftId) (contracts/MeldStakingOperator.sol#208) - stakingStorage.setNodeBaseStakedAmount(nodeId,stakingStorage.getNodeBaseStakedAmount(nodeId) - baseStakedAmount) (contracts/MeldStakingOperator.sol#221-224) - stakingStorage.setNodeLastStakedAmountPerEpoch(nodeId,currentEpoch,nodeLastStakedAmount - stakerLastStakedAmount) (contracts/MeldStakingOperator.sol#229-233) - stakingStorage.setNodeInactive(nodeId) (contracts/MeldStakingOperator.sol#234) - stakingStorage.setTotalBaseStakedAmount(oldTotalBaseStakedAmount - baseStakedAmount) (contracts/MeldStakingOperator.sol#237) - stakingStorage.setLastStakedAmountPerEpoch(currentEpoch,lastStakedAmount - stakerLastStakedAmount) (contracts/MeldStakingOperator.sol#239-242) - stakingStorage.removeStaker(_nftId) (contracts/MeldStakingOperator.sol#245) - stakingCommon.redeemStakingNFT(_nftId) (contracts/MeldStakingOperator.sol#248) - stakingCommon.withdrawMeld(_msgSender(),totalAmount) (contracts/MeldStakingOperator.sol#249) Event emitted after the call(s): - NodeLeft(_nftId,nodeId,totalAmount) (contracts/MeldStakingOperator.sol#255) - RewardsClaimed(_nftId,unclaimedRewards) (contracts/MeldStakingOperator.sol#253) - TotalBaseStakedAmountChanged(_msgSender(),oldTotalBaseStakedAmount,stakingStorage.getTotalBaseStakedAmount()) (contracts/MeldStakingOperator.sol#256-260) | Low |

| Finding | Impact |
|---|---|
| Reentrancy in MeldStakingOperator.requestNode(string,uint256,uint256,uint256,string) (contracts/MeldStakingOperator.sol#123-161): External calls: - stakingStorage.setNodeName(nodeId,_nodeName) (contracts/MeldStakingOperator.sol#146) - nftId = stakingCommon.mintStakingNFT(_msgSender(),_amount) (contracts/MeldStakingOperator.sol#148) - stakingStorage.createNodeRequest(nodeId,nftId,_delegatorFee,_amount,_lockTierId) (contracts/MeldStakingOperator.sol#150) Event emitted after the call(s): - NodeRequestCreated(_msgSender(),nodeId,nftId,_delegatorFee,_amount,_lockTierId,_metadata) (contracts/MeldStakingOperator.sol#152-160) | Low |
| Reentrancy in MeldStakingOperator.cancelNodeRequest(bytes32) (contracts/MeldStakingOperator.sol#168-181): External calls: - stakingStorage.removeNodeRequest(_nodeId) (contracts/MeldStakingOperator.sol#175) - stakingCommon.redeemStakingNFT(operator) (contracts/MeldStakingOperator.sol#177) - stakingCommon.withdrawMeld(_msgSender(),stakingAmount) (contracts/MeldStakingOperator.sol#178) Event emitted after the call(s): - NodeRequestCancelled(_nodeId,operator,stakingAmount) (contracts/MeldStakingOperator.sol#180) | Low |
| MeldStakingOperator.leaveNode(uint256) (contracts/MeldStakingOperator.sol#190-261) uses timestamp for comparisons Dangerous comparisons: - require(bool,string)(block.timestamp >= endOfLocking,STAKING_LOCKED) (contracts/MeldStakingOperator.sol#197) | Low |
| End of table for slither | |

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives.

THANK YOU FOR CHOOSING

// HALBORN