Stader Labs - Hedera Stader Protocol v3

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **Oct 21st, 2022 - Nov 11th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 6
CONTACTS 6
1 EXECUTIVE OVERVIEW 8
1.1 INTRODUCTION 9
1.2 AUDIT SUMMARY 9
1.3 TEST APPROACH & METHODOLOGY 9
RISK METHODOLOGY 10
1.4 SCOPE 12
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 13
3 FINDINGS & TECH DETAILS 14
3.1 (HAL-01) TIMELOCK CAN BE BYPASSED - MEDIUM 16
Description 16
Code Location 16
Risk Level 17
Proof of Concept 18
Recommendation 18
Remediation Plan 19
3.2 (HAL-02) LACK OF TRANSFEROWNERSHIP PATTERN - LOW 20
Description 20
Risk Level 20
Recommendation 20
Remediation Plan 21
3.3 (HAL-03) DIFFERENT PROXYNODE ARRAY LENGTH REQUIREMENTS - LOW 22
Description 22

Risk Level 24
Recommendation 24
Remediation Plan 24
3.4 (HAL-04) MISSING ZERO ADDRESS CHECKS - LOW 25
Description 25
Code Location 25
Risk Level 25
Recommendation 26
Remediation Plan 26
3.5 (HAL-05) LACK OF PARAMETER LIMITS - LOW 27
Description 27
Code Location 27
Risk Level 28
Recommendation 28
Remediation Plan 29
3.6 (HAL-06) MISSING REENTRANCY GUARD - LOW 30
Description 30
Code Location 30
Risk Level 31
Recommendation 31
Remediation Plan 31
3.7 (HAL-07) NODEPROXY ARRAY CANNOT BE MODIFIED - INFORMATIONAL 32
Description 32
Code Location 32
Risk Level 33
Recommendation 33

Remediation Plan 33
3.8 (HAL-08) FLOATING PRAGMA - INFORMATIONAL 34
Description 34
Risk Level 34
Recommendation 34
Remediation Plan 34
3.9 (HAL-09) CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS - INFORMATIONAL 35
Description 35
Code Location 35
Risk Level 36
Recommendation 36
Remediation Plan 36
3.10 (HAL-10) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS - INFORMATIONAL 37
Description 37
Risk Level 37
Recommendation 37
Remediation Plan 37
3.11 (HAL-11) REVERT STRING SIZE OPTIMIZATION - INFORMATIONAL 38
Description 38
Code Location 38
Risk Level 38
Recommendation 38
Remediation Plan 38
3.12 (HAL-12) UNUSED EVENTS - INFORMATIONAL 39
Description 39

Code Location 39
Risk Level 39
Recommendation 40
Remediation Plan 40
3.13 (HAL-13) UNNECESSARY CHECK - INFORMATIONAL 41
Description 41
Code Location 41
Risk Level 42
Recommendation 42
Remediation Plan 42
3.14 (HAL-14) NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES - INFORMATIONAL 43
Description 43
Code Location 43
Risk Level 44
Recommendation 44
Remediation Plan 44
3.15 (HAL-15) USING POSTFIX OPERATORS IN LOOPS - INFORMATIONAL 45
Description 45
Code Location 45
Risk Level 45
Recommendation 45
Remediation Plan 46
3.16 (HAL-16) DIVISION BY ZERO - INFORMATIONAL 47
Description 47
Code Location 47

Risk Level 47
Recommendation 47
Remediation Plan 48
3.17 (HAL-17) SPLITTING REQUIRE() STATEMENTS THAT USES AND OPERATOR SAVES GAS - INFORMATIONAL 49
Description 49
Code Location 49
Proof of Concept 49
Risk Level 50
Recommendation 50
Remediation Plan 50
4 MANUAL TESTING 51
4.1 INTRODUCTION 52
4.2 TESTING 53
REWARD MECHANISM 53
WITHDRAW FUNDS MECHANISM 56
UNDELEGATION MECHANISM 60
5 AUTOMATED TESTING 64
5.1 STATIC ANALYSIS REPORT 65
Description 65
Slither results 65

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/26/2022	Kaan Caglan
0.2	Draft Updates	10/27/2022	Kubilay Onur Gungor
0.3	Draft Updates	10/30/2022	Kaan Caglan
0.4	Draft Updates	11/01/2022	Francisco González
0.5	Draft Review	11/03/2022	Gabi Urrutia
1.0	Remediation Plan	11/04/2022	Francisco González
1.1	Remediation Plan Updates	11/05/2022	Francisco González
1.2	Remediation Plan Review	11/05/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com

Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Kubilay Onur Gungor	Halborn	Kubilay.Gungor@halborn.com
Kaan Caglan	Halborn	Kaan.Caglan@halborn.com
Francisco González	Halborn	Francisco.Villarejo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Stader Labs engaged Halborn to conduct a security audit on their smart contracts beginning on October 21st, 2022 and ending on November 11th, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repositories stader-labs/hedera-stader-protocol-v1

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

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

In summary, Halborn identified some security risks were mostly addressed by the Stader Labs team.

1.3 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 audit. 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 audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Brownie, Remix IDE)

RISK METHODOLOGY:

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

RISK SCALE - LIKELIHOOD

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

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

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

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contract:

- Undelegation.sol
- Timelock.sol
- Staking.sol
- Rewards.sol
- Ownable.sol
- NodeProxy.sol

Audit Commit ID :
- eef82c8d5252f56d3357dd1ba4c1fc788e7faabd

Fixed Commit ID :
- b04fc3be788a6d698071ceb77f6fe844e0ded0e7

Fixed Updated Commit ID:
- c88a979fabb1f1338683d2687155558f7b006b4c

EXECUTIVE OVERVIEW

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	5	11

LIKELIHOOD

IMPACT

(HAL-01)				
(HAL-03)				
(HAL-07)	(HAL-02) (HAL-04) (HAL-05) (HAL-06)			
(HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16) (HAL-17)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - TIMELOCK CAN BE BYPASSED	Medium	SOLVED - 11/05/2022
HAL02 - LACK OF TRANSFEROWNERSHIP PATTERN	Low	SOLVED - 11/05/2022
HAL03 - DIFFERENT PROXYNODE ARRAY LENGTH REQUIREMENTS	Low	SOLVED - 11/04/2022
HAL04 - MISSING ZERO ADDRESS CHECKS	Low	SOLVED - 11/04/2022
HAL05 - LACK OF PARAMETER LIMITS	Low	PARTIALLY SOLVED - 11/04/2022
HAL06 - MISSING REENTRANCY GUARD	Low	SOLVED - 11/04/2022
HAL07 - NODEPROXY ARRAY CANNOT BE MODIFIED	Informational	ACKNOWLEDGED
HAL08 - FLOATING PRAGMA	Informational	SOLVED - 11/04/2022
HAL09 - CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS	Informational	SOLVED - 11/04/2022
HAL10 - USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS	Informational	PARTIALLY SOLVED - 11/05/2022
HAL11 - REVERT STRING SIZE OPTIMIZATION	Informational	ACKNOWLEDGED
HAL12 - UNUSED EVENTS	Informational	SOLVED - 11/04/2022
HAL13 - UNNECESSARY CHECK	Informational	ACKNOWLEDGED
HAL14 - NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES	Informational	ACKNOWLEDGED
HAL15 - USING POSTFIX OPERATORS IN LOOPS	Informational	ACKNOWLEDGED
HAL16 - DIVISION BY ZERO	Informational	SOLVED - 11/04/2022
HAL17 - SPLITTING REQUIRE() STATEMENTS THAT USES AND OPERATOR SAVES GAS	Informational	SOLVED - 11/04/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) TIMELOCK CAN BE BYPASSED - MEDIUM

Description:

Timelock contract is used to queue the transfer of HBAR from the previous Staking contract to the new one. It introduces a lockedPeriod parameter, defining the minimum time between a withdrawal is requested and when it can be completed.

However, it has been detected that the address defined in timelockOwner can either queue funds that could be transferred once lockedPeriod has passed by and also can call setLockedPeriod(), which defines the value of lockedPeriod.

Since the same user who calls queuePartialFunds() or queueAllFunds() can set lockedPeriod by calling setLockedPeriod(), Timelock functionalities can be trivially bypassed by an ill-intentioned user with enough privileges, defeating the whole purpose of the contract.

Code Location:

Listing 1: Timelock.sol (Lines 64,84)

```
61     function queuePartialFunds(address payable to, uint256 amount)
62         external
63         checkZeroAddress(to)
64         checkOwner
65         returns (uint256)
66     {
67         if (amount > address(this).balance) revert("Amount exceeds balance");
68         uint256 index = withdrawQueue.length;
69         Withdraw memory withdrawData = Withdraw({
70             timestamp: block.timestamp,
71             lockedAmount: amount,
72             to: to
73         });
74         withdrawQueue.push(withdrawData);
```

```
        emit Queued(index, amount);
        return index;
    }

    /// @notice queue the transaction for withdrawal of the entire contract balance
    /// @param to address of the account to transfer the tokens to
    function queueAllFunds(address payable to)
        external
        checkZeroAddress(to)
        checkOwner
        returns (uint256)
    {
        uint256 index = withdrawQueue.length;
        Withdraw memory userTransaction = Withdraw({
            timestamp: block.timestamp,
            lockedAmount: address(this).balance,
            to: to
        });
        withdrawQueue.push(userTransaction);
        emit Queued(index, address(this).balance);
        return index;
    }
```

Listing 2: Timelock.sol (Line 146)

```
144     /// @notice Set the locking period for the transfer of tokens
145     /// @param _lockedPeriod time in secs for withholding transfer transaction
146     function setLockedPeriod(uint256 _lockedPeriod) external checkOwner {
147         lockedPeriod = _lockedPeriod;
148     }
```

Risk Level:

Likelihood - 1
Impact - 5

Proof of Concept:

For this Proof of Concept, the user with Owner privileges in Timelock contract will extract immediately the whole balance of Staking contract. To do so, setLockedPeriod() will be called to set a 0 lock period, and then queueAllFunds() and withdraw() will be called consecutively, extracting the complete balance stored in the contract, defeating the purpose of a Time Lock.

```
>>> bypassTimeLocker()

Balance of Staking contract --> 100000000000000000000

Balance of user3 --> 10000000000000000000000000

Current Locked period value --> 7200

Setting 0 Locked period...
Transaction sent: 0x8605a584c8304dca88f4b1f162a545680fe53f0b1705881e5d74ab2ba929736a
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 51
  Staking.setLockedPeriod confirmed   Block: 15884358   Gas used: 13695 (0.00%)


Queuing transfer and immediatly calling withdraw...

contract_Staking.queueAllFunds(user3, {'from': owner}) -->
Transaction sent: 0x22199210a32f150c6ae82ab9e29575505b19977c7e6c8823701d3dec41218116
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 52
  Staking.queueAllFunds confirmed   Block: 15884359   Gas used: 106053 (0.02%)


contract_Staking.withdraw(0, {'from': owner}) -->
Transaction sent: 0x7dd4c71fb51aa219ce3c7aa1c1dc26cb85565e0d7be6f937dee5677e1b4a41ec
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 53
  Staking.withdraw confirmed   Block: 15884360   Gas used: 26791 (0.00%)


Balance of Staking contract --> 0

Balance of user3 --> 10000100000000000000000000
```

Recommendation:

Multiple measures are recommended to enforce the correct use of Timelock:

- Separate the roles in charge of setting lockedPeriod and transferring balances.
- Protect setLockedPeriod() function itself with a Time Lock.
- Add a require statement containing the minimum acceptable value for lockedPeriod.

FINDINGS & TECH DETAILS

If this finding poses no security risk at all, then deleting Timelock contract is recommended for saving gas and using a regular transfer function instead.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by blocking the owner to set the lockedPeriod variable less than 2 days.

3.2 (HAL-02) LACK OF TRANSFEROWNERSHIP PATTERN - LOW

Description:

The current ownership transfer process for Timelock contract involves the current owner calling the setTimeLockOwner() function:

Listing 3: Timelock.sol

```
    /// @notice Set new multisig owner for the transfer of Hbar to
    ↳ new version
    /// @param _timelockOwner the new owner of Hbar withdrawal to
    ↳ new version
    function setTimeLockOwner(address _timelockOwner)
        external
        checkZeroAddress(_timelockOwner)
        checkOwner
    {
        timelockOwner = _timelockOwner;
    }
```

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the checkOwner modifier.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an acceptOwnership() function for the transfer of the ownership to fully succeed. This ensures the nominated **EOA** account is a valid and active account.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by implementing a two-step process.

3.3 (HAL-03) DIFFERENT PROXYNODE ARRAY LENGTH REQUIREMENTS - LOW

Description:

It has been observed that two functions check the length of the input array and compare it to the length of nodeProxyAddresses array in Staking contract, but they do it differently.

collectRewards takes an input array which defines from which contracts rewards will be collected, and a require statement enforces that the length of the array is equal than nodeProxyAddresses length:

Listing 4: Staking.sol (Line 294)

```
286     function collectRewards(uint256[] memory pendingRewardNodeIndexes)
287         external
288         payable
289         whenNotPaused
290         onlyOperator
291     {
292         require(nodeStakingActive, "node staking not active");
293         require(
294             pendingRewardNodeIndexes.length == nodeProxyAddresses.length,
295             "Invalid pendingRewardNodeIndexes input"
296         );
297         for (uint256 i; i < nodeProxyAddresses.length; i++) {
298             if (pendingRewardNodeIndexes[i] == 1) {
299                 require(
300                     address(this).balance >= 1,
301                     "Insufficient balance to execute collectRewards"
302                 );
303                 moveBalanceForStaking(nodeProxyAddresses[i], 1);
304             }
305         }
306     }
```

On the other hand, stakeWithNodes takes as input a similar array in which each index of the position corresponds with each index of the nodeProxyAddresses array. However, this time, the require statement only checks for that array to have the same or lower length than nodeProxyAddresses:

Listing 5: Staking.sol (Line 247)

```
239     function stakeWithNodes(uint256[] calldata amountToSend, uint256 index)
240         external
241         whenNotPaused
242         onlyOperator
243     {
244         require(!nodeStakingActive, "node staking already active");
245         require(index < nodeProxyAddresses.length, "Invalid index");
246         require(
247             amountToSend.length <= nodeProxyAddresses.length,
248             "Invalid size of amountToSend"
249         );
250         nodeStakingActive = true;
251         balanceBefore = address(this).balance;
252         // iterating over amountToSend array to send hbar to respective index of nodeProxyContract
253         // following checks are to incorporate changes in the staking contract balance after computing amountToSend
254         for (uint256 i = 0; i < amountToSend.length; i++) {
255             if (
256                 amountToSend[i] > 0 &&
257                 address(this).balance > 0 &&
258                 address(this).balance >= amountToSend[i]
259             ) {
260                 moveBalanceForStaking(nodeProxyAddresses[i], amountToSend[i]);
261             } else if (
262                 amountToSend[i] > 0 &&
263                 address(this).balance > 0 &&
264                 amountToSend[i] > address(this).balance
265             ) {
266                 moveBalanceForStaking(
267                     nodeProxyAddresses[i],
268                     address(this).balance
```

```
                    );
                }
            }
            if (address(this).balance > 0) {
                moveBalanceForStaking(
                    nodeProxyAddresses[index],
                    address(this).balance
                );
            }
            emit stakedWithNodes(balanceBefore);
        }

```

Having two arrays in which each position corresponds with the same position of nodeProxyAddresses array but with different length requirements might be confusing and error-prone, since the absence of a single position in amountToSend could mean that every amount defined in the array is staked into the wrong node.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to unify the criteria of the require statements on both functions, enforcing that amountToSend array has the same length of nodeProxyAddresses to prevent confusions or any input error.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by requiring amountToSend to have the same length as nodeProxyAddresses.

3.4 (HAL-04) MISSING ZERO ADDRESS CHECKS - LOW

Description:

The constructor of the Rewards.sol contract is missing address validation. Every address should be validated and checked that it is different from zero. Control of that constructor is wrong because of the or statement between them. Because of that issue, either the _stakerAddress variable or _daoAddress might be a 0 address, and it can cause an unintended loss in the distributeStakingRewards function. This is also considered a best practice.

Code Location:

Listing 6: Rewards.sol (Line 61)

```
59     constructor(address payable _stakerAddress, address payable
  ↳ _daoAddress) {
60         require(
61             _stakerAddress != address(0) || _daoAddress != address
  ↳ (0),
62             "Address cannot be a zero"
63         );
64         stakerAddress = _stakerAddress;
65         daoAddress = _daoAddress;
66         genesisTimestamp = block.timestamp;
67         lastRedeemedTimestamp = genesisTimestamp;
68         // _pause();
69     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

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

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by enforcing separate zero address checks on each of the input addresses.

3.5 (HAL-05) LACK OF PARAMETER LIMITS - LOW

Description:

It has been detected that some parameter modifying functions do not have logical limits. This may cause the contract to function with parameter values that, although allowed, make no sense in the context of the application, which might cause a variety of problems or even rendering the contract unusable.

There are two functions in Undelegation.sol and Timelock.sol contracts that should have a minimum value check in place. These functions determine the minimum time needed for being able to unstake **HBAR** from Staking contract and for transferring funds from the old staking contract to the new one, respectively.

Having no minimum value check means that **HBAR** could be immediately unstaked or Timelock could be bypassed.

Similarly, it is recommended to define some boundaries on Staking.sol's minDeposit and maxDeposit, since setting a minDeposit value too high or a maxDeposit value too low (or zero) would prevent anyone from being able to stake HBAR.

Code Location:

Listing 7: "Timelock.sol

```
    function setLockedPeriod(uint256 _lockedPeriod) external
 ↳ checkOwner {
        lockedPeriod = _lockedPeriod;
    }
```

Listing 8: "Undelegation.sol

```
    function setUnbondingTime(uint256 _unbondingTime) external onlyOwner {
        unbondingTime = _unbondingTime;
        emit NewUnbondingTime(unbondingTime);
    }
```

Listing 9: "Staking.sol

```
    /// @notice Set minimum deposit amount (onlyOwner)
    /// @param _newMinDeposit the minimum deposit amount in multiples of 10**8
    function updateMinDeposit(uint256 _newMinDeposit) external onlyOwner {
        minDeposit = _newMinDeposit;
    }

    /// @notice Set maximum deposit amount (onlyOwner)
    /// @param _newMaxDeposit the maximum deposit amount in multiples of 10**8
    function updateMaxDeposit(uint256 _newMaxDeposit) external onlyOwner {
        maxDeposit = _newMaxDeposit;
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to enforce logical value limits for critical parameters and check for additional occurrences of this same vulnerability.

Remediation Plan:

PARTIALLY SOLVED: The Stader Labs team partially solved this finding by adding some logical checks on the Staking.sol contract, enforcing that minDeposit is lower than maxDeposit, and maxDeposit is greater than minDeposit.

3.6 (HAL-06) MISSING REENTRANCY GUARD - LOW

Description:

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

Code Location:

Listing 10: Timelock.sol

```
    function withdraw(uint256 index) external returns (uint256) {
        if (address(this).balance == 0) revert("No funds to withdraw");
        if (index >= withdrawQueue.length) revert("Invalid index");
        Withdraw storage withdrawData = withdrawQueue[index];
        if (withdrawData.timestamp + lockedPeriod >= block.timestamp)
            revert("Unlock period not expired");
        if (withdrawData.lockedAmount == 0) revert("Amount not available");
        address payable to = withdrawData.to;
        uint256 amount = withdrawData.lockedAmount;
        delete withdrawQueue[index];
        // payable(to).transfer(amount);
        Address.sendValue(payable(to), amount);
        emit Transferred(index, amount, to);
        return index;
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

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

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by adding the nonReentrant modifier to the withdraw() function on the Timelock.sol contract.

3.7 (HAL-07) NODEPROXY ARRAY CANNOT BE MODIFIED - INFORMATIONAL

Description:

For each node in Hedera network, a NodeProxy contract will be deployed. Each one of these contracts is assigned to one node, and staking on one of the contracts will effectively mean the same as staking on the node itself.

The addresses of these contracts are stored in nodeProxyAddresses array, set in Staking contract's constructor function.

However, there is no way to modify the addresses contained in the array, so any eventuality related with Hedera nodes might potentially render Staking contract partially or even completely unusable. These eventualities might be related to node additions, node ID changes, etc.

Code Location:

Listing 11: Staking.sol (Line 110)

```
104     constructor(
105         address _hbarxAddress,
106         address _multisigAdminAddress,
107         address payable _undelegationContractAddress,
108         uint256 _totalSupply,
109         address _operator,
110         address[] memory _nodeProxyAddresses
111     )
112         Timelock(_multisigAdminAddress)
113         checkZeroAddress(_hbarxAddress)
114         checkZeroAddress(_undelegationContractAddress)
115         checkZeroAddress(_operator)
116     {
117         hbarxAddress = _hbarxAddress;
118         undelegationContractAddress = _undelegationContractAddress
    ↳ ;
119         totalSupply = _totalSupply;
```

```
        operator = _operator;
        for (uint256 i = 0; i < _nodeProxyAddresses.length; i++) {
            if (_nodeProxyAddresses[i] == address(0))
                revert("zero address for nodeProxy");
            nodeProxyAddresses.push(payable(_nodeProxyAddresses[i
 ↳ ]));
        }
    }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to implement a protected function allowing Stader Labs team to modify nodeProxyAddresses array, adding flexibility in the eventuality of any node change on Hedera network.

Remediation Plan:

ACKNOWLEDGED: The Stader Labs team acknowledged this finding.

3.8 (HAL-08) FLOATING PRAGMA - INFORMATIONAL

Description:

Hedera Stader Protocol contracts use floating pragma. Contracts should be deployed with the same compiler version and flags they have tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version that is too new and has not been extensively tested.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider locking the pragma version with known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by fixing the pragma version for each contract in scope.

3.9 (HAL-09) CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS - INFORMATIONAL

Description:

Reading array length at each iteration of the loop takes 6 gas (3 for mload and 3 to place memory_offset) in the stack. Caching the array length in the stack saves around 3 gas per iteration.

Code Location:

Listing 12: Staking.sol (Lines 294,297)

```
286     function collectRewards(uint256[] memory
    ↳ pendingRewardNodeIndexes)
287         external
288         payable
289         whenNotPaused
290         onlyOperator
291     {
292         require(nodeStakingActive, "node staking not active");
293         require(
294             pendingRewardNodeIndexes.length == nodeProxyAddresses.
    ↳ length,
295             "Invalid pendingRewardNodeIndexes input"
296         );
297         for (uint256 i; i < nodeProxyAddresses.length; i++) {
```

Listing 13: Staking.sol (Line 314)

```
312     function withdrawFromNodes() external whenNotPaused
    ↳ onlyOperator {
313         require(nodeStakingActive, "node staking not active");
314         for (uint256 i; i < nodeProxyAddresses.length; i++) {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider the cache array length.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by caching the array length when needed.

3.10 (HAL-10) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS - INFORMATIONAL

Description:

Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. If the revert string uses strings to provide additional information about failures (e.g. `require(!isStakePaused, 'Staking is paused');`), but they are rather expensive, especially when it comes to deploying cost, and it is difficult to use dynamic information in them.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to implement custom errors instead of revert strings.

Remediation Plan:

PARTIALLY SOLVED: The Stader Labs team partially solved this finding by changing revert strings to custom errors in a few files.

3.11 (HAL-11) REVERT STRING SIZE OPTIMIZATION - INFORMATIONAL

Description:

Shortening the revert strings to fit within 32 bytes will decrease deployment time gas and decrease runtime gas when the revert condition is met.

Revert strings that are longer than 32 bytes require at least one additional mstore, along with additional overhead to calculate memory offset, etc. For example:

Code Location:

Listing 14: Rewards.sol

```
        require(
            address(this).balance > 0,
            "Contract balance is should be greater than 0"
        );
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Shorten the revert strings to fit within 32 bytes. That will affect gas optimization.

Remediation Plan:

ACKNOWLEDGED: The Stader Labs team acknowledged this finding.

3.12 (HAL-12) UNUSED EVENTS - INFORMATIONAL

Description:

The following events are declared, but they are not emitted by any function:

Code Location:

NodeProxy.sol
- Line 17:
```
event Received(address indexed from, uint256 amount);
```
- Line 20:
```
event Fallback(address indexed from, uint256 amount);
```
- Line 29:
```
event CollectedRewards();
```

Staking.sol
- Line 70:
```
event Undelegated(address indexed to, uint256 amount);
```

Undelegation.sol
- Line 27:
```
event Received(address from, uint256 amount);
```
- Line 29:
```
event Fallback(address from, uint256 amount);
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Check whether these events should be used and if not remove them.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by removing unnecessary events.

3.13 (HAL-13) UNNECESSARY CHECK - INFORMATIONAL

Description:

The distributeStakingRewards function under Rewards.sol does check if the daoFeesPercentage is less than 100 and then reverts. However, that condition will never be reachable as the setDaoFeesPercentage function guarantees that daoFeesPercentage won't be able to higher or equal to 100.

Code Location:

Listing 15: Rewards.sol (Lines 86,87,88,89)

```
81     function distributeStakingRewards() external whenNotPaused
   ↳ nonReentrant {
82         require(
83             address(this).balance > 0,
84             "Contract balance is should be greater than 0"
85         );
86         require(
87             daoFeesPercentage < 100,
88             "Dao fees percentage should be less than 100"
89         );
```

Listing 16: Rewards.sol (Line 151)

```
146     function setDaoFeesPercentage(uint256 _daoFeesPercentage)
147         external
148         onlyOwner
149     {
150         require(
151             _daoFeesPercentage < 100,
152             "Dao fees percentage should be less than 100"
153         );
154         daoFeesPercentage = _daoFeesPercentage;
155     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove unnecessary checks to reduce gas costs

Remediation Plan:

ACKNOWLEDGED: The Stader Labs team acknowledged this finding.

3.14 (HAL-14) NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES - INFORMATIONAL

Description:

uint256 variables are already initialized to 0 by default. uint256 public epoch = 0 would reassign the 0 to epoch which wastes gas.

The same occurs with bool and address variables. They are already initialized to false/address(0).

Code Location:

Rewards.sol
- Line 21:

```
uint256 public epoch = 0;
```

Staking.sol
- Line 30:

```
bool public isStakePaused = false;
```

- Line 31:

```
bool public isUnstakePaused = false;
```

- Line 32:

```
bool public nodeStakingActive = false;
```

- Line 36:

```
uint256 public minDeposit = 0;
```

- Line 39:

```
uint256 public totalSupply = 0;
```

- Line 121:

```
for (uint256 i = 0; i < _nodeProxyAddresses.length; i++){
```

- Line 254:

```
for (uint256 i = 0; i < amountToSend.length; i++){
```

NodeProxy.sol
- Line 14:

```
address payable public stakerAddress = payable(address(0));
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not initialize uint variables to 0. bool variables to false and address variables to address(0) to save some gas. For example, use instead:

```
uint256 public totalSupply;
```

Remediation Plan:

ACKNOWLEDGED: The Stader Labs team acknowledged this finding.

3.15 (HAL-15) USING POSTFIX OPERATORS IN LOOPS - INFORMATIONAL

Description:

In the loops below, postfix operators (e.g. `i++`) were used to increment or decrement the value of variables. In loops, using prefix operators (e.g., `++i`) costs less gas per iteration than postfix operators.

Code Location:

Staking.sol

- Line 121:

```
for (uint256 i = 0; i < _nodeProxyAddresses.length; i++){
```

- Line 254:

```
for (uint256 i = 0; i < amountToSend.length; i++){
```

- Line 297:

```
for (uint256 i; i < nodeProxyAddresses.length; i++){
```

- Line 314:

```
for (uint256 i; i < nodeProxyAddresses.length; i++){
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i++` to increment the value of an `uint` variable inside a loop. This does not only apply to the iterator variable. It also applies to the increments/decrements done inside the loop code block.

Remediation Plan:

ACKNOWLEDGED: The Stader Labs team acknowledged this finding.

3.16 (HAL-16) DIVISION BY ZERO - INFORMATIONAL

Description:

Calling the getExchangeRate function with totalSupply as 0 and nodeStakingActive as true, will cause the function to throw a division by zero error.

Code Location:

Listing 17: "Staking.sol (Lines 367,368)

```
365     function getExchangeRate() external view returns (uint256) {
366         ///@dev 1HBar = 100_000_000 tinybar
367         if (nodeStakingActive) {
368             return (balanceBefore * decimals) / totalSupply;
369         }
370         uint256 exchangeRate = 1 * decimals;
371         if (totalSupply == 0 || address(this).balance == 0) {
372             return exchangeRate;
373         } else {
374             exchangeRate = ((address(this).balance) * decimals) /
    ↳ totalSupply;
375         }
376         return exchangeRate;
377     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Make sure to validate all operands used during a math operation and inform the user of unappropriated state by reverting the transaction with a custom message.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by validating all operands before performing math operations.

3.17 (HAL-17) SPLITTING REQUIRE() STATEMENTS THAT USES AND OPERATOR SAVES GAS - INFORMATIONAL

Description:

Instead of using the && operator in a single require statement to check multiple conditions, using multiple require statements with one condition per require statement will save 8 GAS per &&
The gas difference would only be realized if the revert condition is realized (met).

Code Location:

Listing 18: Staking.sol

```
        require(
            hbarReceived > minDeposit && hbarReceived <= maxDeposit,
            "Deposit amount must be within valid range"
        );
```

Listing 19: Undelegation.sol

```
        require(
            undelegateData.amount != 0 && undelegateData.timestamp != 0,
            "Undelegation not found"
        );
```

Proof of Concept:

The following tests were carried out in remix with both optimization turned on and off

Listing 20

```
require ( a > 1 && a < 5, "Initialized");
return  a + 2;
```

Execution cost
21617 with optimization and using &&
21976 without optimization and using &&

After splitting the require statement

Listing 21

```
require (a > 1 ,"Initialized");
require (a < 5 , "Initialized");
return a + 2;
```

Execution cost
21609 with optimization and split require
21968 without optimization and using split require

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

For best security practices, consider as much as possible, declaring events at the end of the function. Events can be used to detect the end of the operation.

Remediation Plan:

SOLVED: The Stader Labs team solved this finding by splitting require statements.

MANUAL TESTING

4.1 INTRODUCTION

Halborn performed different manual tests in all the different Facets of the **Hedera protocol**, trying to find any logic flaws and vulnerabilities.

During the manual tests, the following areas were reviewed carefully :

1. Reward mechanism.
2. Withdraw funds mechanism.
3. Undelegation mechanism.

4.2 TESTING

REWARD MECHANISM:

In the Rewards.sol contract there is a function named distributeStakingRewards and this function is responsible to distribute staking rewards among those with staker address and DAO address.

Listing 22: Rewards.sol

```
    function distributeStakingRewards() external whenNotPaused nonReentrant {
        require(
            address(this).balance > 0,
            "Contract balance is should be greater than 0"
        );
        require(
            daoFeesPercentage < 100,
            "Dao fees percentage should be less than 100"
        );
        uint256 currentTimestamp = block.timestamp;
        uint256 epochDelta = (currentTimestamp - lastRedeemedTimestamp);
        lastRedeemedTimestamp = currentTimestamp;
        epoch++;
        uint256 epochRewards = (epochDelta * emissionRate);

        uint256 totalRewards = address(this).balance;
        if (epochRewards > totalRewards) epochRewards = totalRewards; // this is important

        uint256 daoFees = (epochRewards * daoFeesPercentage) / 100;

        // payable(stakerAddress).transfer(epochRewards - daoFees);
        Address.sendValue(payable(stakerAddress), epochRewards - daoFees);
        emit DistributedRewards(
            stakerAddress,
            epochRewards - daoFees,
            currentTimestamp
        );
```

```
        // payable(daoAddress).transfer(daoFees);
        Address.sendValue(payable(daoAddress), daoFees);
        emit DaoTransfer(daoAddress, daoFees, currentTimestamp);
    }
```

This function does not have any kind of `msg.sender` control, So anyone would be able to call this function. This function distributes rewards every 24 hours. A malicious actor can call this function before 24 hours, but a malicious actor can call this anytime. However, he will not be able to manipulate this function somehow because of the correctness of `totalRewards` and `daoFees` calculations. Even malicious actor calls this in 12 hours, rewards will be the same because it always calculates it with `lastRedemeedTiestamp` variable.

```
>>> lastRedeemedTimestamp = rewardsContract.lastRedeemedTimestamp()
>>> stakingContract.balance()
0
>>> rewardsContract.balance()
1000000000000000000
>>> tx1 = rewardsContract.distributeStakingRewards({'from': user1})
Transaction sent: 0x5cf4ea76175d8a4e62494e1afc08d4fc68341ba096b587937e5f0b4d13b42bd6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Rewards.distributeStakingRewards confirmed   Block: 15   Gas used: 77636 (0.65%)

>>> stakingContract.balance()
83516507676
>>> epochDelta = tx1.timestamp - lastRedeemedTimestamp
>>> secondLastRedeemedTimestamp = rewardsContract.lastRedeemedTimestamp()
>>> tx2 = rewardsContract.distributeStakingRewards({'from': user1})
Transaction sent: 0xc49a2ee762d3ed47da0535b0df8c3728219b6b375032112e82d1971b9cf8c6aa
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  Rewards.distributeStakingRewards confirmed   Block: 16   Gas used: 62636 (0.52%)

>>> stakingContract.balance()
272820591742
```

```
>>> epochDeltaShouldBe = tx2.timestamp - lastRedeemedTimestamp
>>> secondEpochDelta = tx2.timestamp - secondLastRedeemedTimestamp
>>> epochDeltaShouldBe == secondEpochDelta + epochDelta
True
>>> epochRewards = epochDeltaShouldBe * rewardsContract.emissionRate()
>>> initialRewardsBalance = 1000000000000000000
>>> if epochRewards > initialRewardsBalance:
...     epochRewards = initialRewardsBalance
...
>>> daoFees = epochRewards * rewardsContract.daoFeesPercentage() / 100
>>> stakerShouldGet = epochRewards - daoFees
```

Listing 23

```
>>> stakerShouldGet
272820591741.6
>>> stakingContract.balance()
```

```
272820591742
```

So even attacker calls this function 2 times, in the end staker gets the same balance because the math is correct. The only difference here is the precision loss in division operation in solidity. The only way to exploit this function is to make `currentTimestamp` variable to be equal to `lastRedeemedTimestamp` and if an attacker can do that. `epochDelta` will be 0 and eventually staker or DAO won't be able to get any rewards. But it's not likely possible because the attacker can't send thousands of transactions to make it pass in the same block time. The average block time mining in Ethereum is 12 seconds, so the attacker won't be able to do this.

WITHDRAW FUNDS MECHANISM:

In the Timelock.sol contract there is a function named withdraw, with using this function users can withdraw their funds. And there are two other functions named queuePartialFunds and queueAllFunds which allow the owner to queue funds.

Listing 24: Timelock.sol

```
    function queuePartialFunds(address payable to, uint256 amount)
        external
        checkZeroAddress(to)
        checkOwner
        returns (uint256)
    {
        if (amount > address(this).balance) revert("Amount exceeds balance");
        uint256 index = withdrawQueue.length;
        Withdraw memory withdrawData = Withdraw({
            timestamp: block.timestamp,
            lockedAmount: amount,
            to: to
        });
        withdrawQueue.push(withdrawData);
        emit Queued(index, amount);
        return index;
    }
```

Listing 25: Timelock.sol

```
    function queueAllFunds(address payable to)
        external
        checkZeroAddress(to)
        checkOwner
        returns (uint256)
    {
        uint256 index = withdrawQueue.length;
        Withdraw memory userTransaction = Withdraw({
            timestamp: block.timestamp,
            lockedAmount: address(this).balance,
            to: to
        });
        withdrawQueue.push(userTransaction);
```

```
94         emit Queued(index, address(this).balance);
95         return index;
96     }
```

Listing 26: Timelock.sol, (Lines 109,111)

```
100     function withdraw(uint256 index) external returns (uint256) {
101         if (address(this).balance == 0) revert("No funds to withdraw");
102         if (index >= withdrawQueue.length) revert("Invalid index");
103         Withdraw storage withdrawData = withdrawQueue[index];
104         if (withdrawData.timestamp + lockedPeriod >= block.timestamp)
105             revert("Unlock period not expired");
106         if (withdrawData.lockedAmount == 0) revert("Amount not available");
107         address payable to = withdrawData.to;
108         uint256 amount = withdrawData.lockedAmount;
109         delete withdrawQueue[index];
110         // payable(to).transfer(amount);
111         Address.sendValue(payable(to), amount);
112         emit Transferred(index, amount, to);
113         return index;
114     }
```

At the end of the withdraw function, contract is sending withdrawData.lockedAmount amount to withdrawData.to user. Address.sendValue is sending the given Ethereum amount to the user with .call function.

Listing 27: Address.sol, (Line 63)

```
60     function sendValue(address payable recipient, uint256 amount) internal {
61         require(address(this).balance >= amount, "Address: insufficient balance");
62 
63         (bool success, ) = recipient.call{value: amount}("");
64         require(success, "Address: unable to send value, recipient may have reverted");
65     }
```

So in theory it is possible to create another contract to make a re-entrancy attack because withdraw function does not have any re-entrancy guard mechanism. However, withdraw function follows the Check-Effects -Interaction pattern correctly because it is deleting the index before sending the Ethereum to the user. So, it is not likely possible to do re-entracy attack in this function. There is no msg.sender control in this withdraw function, so that means any user can call this withdraw function with any parameter and withdraw funds for someone else’s Ethereum to their address, but it is not a vulnerability because at the end the correct user is funded.

```
NameError: name 'clear' is not defined
>>> stakingContract.queuePartialFunds(user1, web3.toWei('0.1', 'ether'), {'from': deployer})
Transaction sent: 0x22a314a69c0fdc126343f922593b70e85b9d44a09372908d23d4b6799ec56a49
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  Staking.queuePartialFunds confirmed   Block: 19   Gas used: 106354 (0.89%)

<Transaction '0x22a314a69c0fdc126343f922593b70e85b9d44a09372908d23d4b6799ec56a49'>
>>> stakingContract.queuePartialFunds(user2, web3.toWei('0.1', 'ether'), {'from': deployer})
Transaction sent: 0x0c752e8b96cb0eacc7a63bc0f66773fa695b7849efd375695a3e5dfcb879b009
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  Staking.queuePartialFunds confirmed   Block: 20   Gas used: 91342 (0.76%)

<Transaction '0x0c752e8b96cb0eacc7a63bc0f66773fa695b7849efd375695a3e5dfcb879b009'>
>>> stakingContract.withdrawQueue(0)
(1667128388, 100000000000000000, "0x33A4622B82D4c04a53e170c638B944ce27cffce3")
>>> stakingContract.withdrawQueue(1)
(1667128391, 100000000000000000, "0x0063046686E46Dc6F15918b61AE2B121458534a5")
>>> stakingContract.lockedPeriod()
7200
>>> stakingContract.setLockedPeriod(0, {'from': deployer})
Transaction sent: 0x9cfd78d1b30f7f617bb26e14c76638916def940d94eed6f00d2ece6d2ddbdf63
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  Staking.setLockedPeriod confirmed   Block: 21   Gas used: 13695 (0.11%)

<Transaction '0x9cfd78d1b30f7f617bb26e14c76638916def940d94eed6f00d2ece6d2ddbdf63'>
>>> stakingContract.lockedPeriod()
0
```

```
>>> user1.balance()
100000000000000000000
>>> user2.balance()
100000000000000000000
>>> stakingContract.withdraw(0, {'from': user1})
Transaction sent: 0x654fb22e95687c073d770f273ee86caf486c940bd5a17e8749d4cc7d8531bbb5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  Staking.withdraw confirmed   Block: 22   Gas used: 26791 (0.22%)

<Transaction '0x654fb22e95687c073d770f273ee86caf486c940bd5a17e8749d4cc7d8531bbb5'>
>>> user1.balance()
100010000000000000000
>>> stakingContract.withdraw(0, {'from': user1})
Transaction sent: 0xda7ad0553cad49830fd52925616853c594ac2d46d11181a00ad28a813dcbc00e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  Staking.withdraw confirmed (Amount not available)   Block: 23   Gas used: 25905 (0.22%)

<Transaction '0xda7ad0553cad49830fd52925616853c594ac2d46d11181a00ad28a813dcbc00e'>
>>> stakingContract.withdraw(1, {'from': user1})
Transaction sent: 0x032af0b94cda6f680d2910e5a32ca5b665441095ac0e7eaf2574ff34d1b6bcf2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  Staking.withdraw confirmed   Block: 24   Gas used: 26797 (0.22%)

<Transaction '0x032af0b94cda6f680d2910e5a32ca5b665441095ac0e7eaf2574ff34d1b6bcf2'>
>>> user1.balance()
100010000000000000000
>>> user2.balance()
100010000000000000000
>>>
```

UNDELEGATION MECHANISM:

In the Staking.sol contract after users unstake their **HBARX** to withdraw their money, unStake function calls the undelegate function of Undelegation.sol contract.

Listing 28: Staking.sol, (Lines 222,223,224)

```
function unStake(uint256 amount) external whenNotPaused returns (uint256) {
        require(!nodeStakingActive, "node Staking is active");
        require(!isUnstakePaused, "Unstaking is paused");
        uint256 hbarxToBurn = (amount);

        uint256 hbarToTransfer = hbarxToBurn; // exchange rate = 1
        if (totalSupply != 0) {
            hbarToTransfer =
                (hbarxToBurn * ((address(this).balance))) /
                (totalSupply);
        }

        ///@dev transfer hbarx to the provided address
        int256 transferTokenResponse = HederaTokenService.transferToken(
            hbarxAddress,
            msg.sender,
            address(this),
            hbarxToBurn.toInt256().toInt64()
        );

        if (transferTokenResponse != HederaResponseCodes.SUCCESS) {
            revert("HBARX transfer failed");
        }

        ///@dev burn hbarx tokens
        (int256 burnTokenResponse, uint64 newTotalSupply) = HederaTokenService
            .burnToken(hbarxAddress, hbarxToBurn.toUint64(), new int64[](0));
        totalSupply = uint256(newTotalSupply);
        if (burnTokenResponse != HederaResponseCodes.SUCCESS) {
            revert("HBARX burn failed");
        }
```

```

        ///@dev move tokens to undelegation contract
        (bool success, ) = payable(undelegationContractAddress).call{
            value: hbarToTransfer
        }(abi.encodeWithSignature("undelegate(address)", msg.sender));
        if (!success) {
            revert("Transfer failed");
        }
        emit UnStaked(msg.sender, hbarToTransfer, hbarxToBurn);
        ///@dev return hbars for transaction
        return hbarToTransfer;
    }
```

The only way to call undelegate function is by calling the unStake function because undelegate function is checking if msg.sender is stakingContractAddress.

Listing 29: Undelegation.sol, (Lines 44,45,46,47,49)

```
42    function undelegate(address to) external payable returns (uint256) {
43        require(msg.value > 0, "Undelegate amount must be greater than 0");
44        require(
45            msg.sender == stakingContractAddress,
46            "Only staking contract can undelegate"
47        );
48
49        undelegationsMap[to].push(Undelegate(block.timestamp, msg.value));
50        emit Undelegated(to, msg.value);
51        return msg.value;
52    }
```

And this function pushes given msg.value and to parameters to undelegationsMap for each user. After that step, users can call withdraw function to withdraw their money. withdraw function is calling Address.sendValue like in Timelock contract. However, in this case, there is a nonReentrant modifier to block the user to make re-entrancy attacks. Even without nonReentrant guard also on the function Check-Effects-Interaction pattern is used correctly.

Listing 30: Undelegation.sol, (Lines 60,72,74)

```
60     function withdraw(uint256 index) external whenNotPaused nonReentrant {
61         Undelegate storage undelegateData = undelegationsMap[msg.sender][index];
62         require(
63             undelegateData.amount != 0 && undelegateData.timestamp != 0,
64             "Undelegation not found"
65         );
66         require(
67             undelegateData.timestamp + unbondingTime <= block.timestamp,
68             "Release time not reached"
69         );
70 
71         uint256 amount = undelegateData.amount;
72         delete undelegationsMap[msg.sender][index];
73         // payable(msg.sender).transfer(amount);
74         Address.sendValue(payable(msg.sender), amount);
75         emit Withdrawn(msg.sender, amount);
76     }
```

So, it is not possible to make re-entrancy attacks or get more money than you also deserve in this function.

```
>>> undelegationContract.undelegate(user2, {'from': stakingContract, 'value': web3.toWei('0.1', 'ether')})
Transaction sent: 0x6e47dd21c9bb4ebe365cf16b6e54bfae50030ed3b10de14ba1455e2837caf276
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  Undelegation.undelegate confirmed   Block: 27   Gas used: 85184 (0.71%)

<Transaction '0x6e47dd21c9bb4ebe365cf16b6e54bfae50030ed3b10de14ba1455e2837caf276'>
>>> undelegationContract.setUnbondingTime(0, {'from': deployer})
Transaction sent: 0x6771742e98ba93eea7796b28fb57ef849a097e3f9678bc5b56ce1d23581c09c4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 20
  Undelegation.setUnbondingTime confirmed   Block: 28   Gas used: 24210 (0.20%)

<Transaction '0x6771742e98ba93eea7796b28fb57ef849a097e3f9678bc5b56ce1d23581c09c4'>
>>> undelegationContract.undelegationsMap(user2, 0)
(1667132869, 100000000000000000)
>>> undelegationContract.unbondingTime()
0
>>> user2.balance()
1000100000000000000000
>>> undelegationContract.withdraw(0, {'from': user2})
Transaction sent: 0x7d7b6a47bab9a7f7c72910dc7aa56f26e4ebd85102889c9a6edd1b2466c86a94
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Undelegation.withdraw confirmed   Block: 29   Gas used: 27053 (0.23%)

<Transaction '0x7d7b6a47bab9a7f7c72910dc7aa56f26e4ebd85102889c9a6edd1b2466c86a94'>
>>> user2.balance()
1000200000000000000000
>>> undelegationContract.withdraw(0, {'from': user2})
Transaction sent: 0xb148338f08afc347cfe1b590e51a78a75ec1d468c01e6f1a60f8ffdeee7f557f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  Undelegation.withdraw confirmed (Undelegation not found)   Block: 30   Gas used: 30066 (0.25%)

<Transaction '0xb148338f08afc347cfe1b590e51a78a75ec1d468c01e6f1a60f8ffdeee7f557f'>
>>> undelegationContract.undelegationsMap(user2, 0)
(0, 0)
>>> 
```

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repositories and was able to compile them correctly into their ABIs and binary format, Slither was run against the 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:

```
NodeProxy.lockStakingContract(address) (contracts/NodeProxy.sol#78-85) should emit an event for:
        - stakingContractId = _stakingContractId (contracts/NodeProxy.sol#84)
Timelock.setTimeLockOwner(address) (contracts/Timelock.sol#126-132) should emit an event for:
        - timelockOwner = _timelockOwner (contracts/Timelock.sol#131)
Staking.updateOperatorAddress(address) (contracts/Staking.sol#347-349) should emit an event for:
        - operator = _operator (contracts/Staking.sol#348)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

Rewards.setDaoFeesPercentage(uint256) (contracts/Rewards.sol#124-127) should emit an event for:
        - daoFeesPercentage = _daoFeesPercentage (contracts/Rewards.sol#126)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Staking.transferToNode(address,uint256) (contracts/Staking.sol#299-302) has external calls inside a loop: (success) = (_nodeProxy).call{value: amount}(abi.encodeWithSignature(receiveFunds())) (contr
acts/Staking.sol#300)
Staking.collectRewards() (contracts/Staking.sol#253-264) has external calls inside a loop: (success) = (nodeProxyAddresses[i]).call(abi.encodeWithSignature(collectRewards())) (contracts/Staking.sol#
257-258)
Staking.withdrawFromNodes() (contracts/Staking.sol#270-296) has external calls inside a loop: (transferFundSuccess) = (nodeProxyAddresses[i]).call(abi.encodeWithSignature(transferFund())) (contracts
/Staking.sol#274-276)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```

```
Different versions of Solidity are used:
        - Version used: ['>=0.4.9<0.9.0', '>=0.5.0<0.9.0', '^0.8.0', '^0.8.1', '^0.8.9']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/ReentrancyGuard.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SafeCast.sol#4)
        - >=0.4.9<0.9.0 (contracts/HederaResponseCodes.sol#2)
        - >=0.5.0<0.9.0 (contracts/HederaTokenService.sol#2)
        - ABIEncoderV2 (contracts/HederaTokenService.sol#3)
        - >=0.4.9<0.9.0 (contracts/IHederaTokenService.sol#2)
        - ABIEncoderV2 (contracts/IHederaTokenService.sol#3)
        - ^0.8.9 (contracts/NodeProxy.sol#1)
        - ^0.8.0 (contracts/Ownable.sol#4)
        - ^0.8.9 (contracts/Rewards.sol#2)
        - ^0.8.9 (contracts/Staking.sol#2)
        - ^0.8.9 (contracts/Timelock.sol#2)
        - ^0.8.9 (contracts/Undelegation.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

HederaTokenService.associateTokens(address,address[]) (contracts/HederaTokenService.sol#94-102) is never used and should be removed
HederaTokenService.cryptoTransfer(IHederaTokenService.TokenTransferList[]) (contracts/HederaTokenService.sol#14-22) is never used and should be removed
HederaTokenService.dissociateToken(address,address) (contracts/HederaTokenService.sol#139-144) is never used and should be removed
HederaTokenService.dissociateTokens(address,address[]) (contracts/HederaTokenService.sol#129-137) is never used and should be removed
HederaTokenService.transferNFT(address,address,address,int64) (contracts/HederaTokenService.sol#225-241) is never used and should be removed
HederaTokenService.transferNFTs(address,address[],address[],int64[]) (contracts/HederaTokenService.sol#175-191) is never used and should be removed
HederaTokenService.transferTokens(address,address[],int64[]) (contracts/HederaTokenService.sol#154-168) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/ReentrancyGuard.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SafeCast.sol#4) allows old versions
Pragma version>=0.4.9<0.9.0 (contracts/HederaResponseCodes.sol#2) is too complex
Pragma version>=0.5.0<0.9.0 (contracts/HederaTokenService.sol#2) is too complex
Pragma version>=0.4.9<0.9.0 (contracts/IHederaTokenService.sol#2) is too complex
Pragma version^0.8.9 (contracts/NodeProxy.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (contracts/Ownable.sol#4) allows old versions
Pragma version^0.8.9 (contracts/Rewards.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/Staking.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/Timelock.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/Undelegation.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

AUTOMATED TESTING

```
Parameter Rewards.setEmissionRate(uint256)._emissionRate (contracts/Rewards.sol#97) is not in mixedCase
Parameter Rewards.setStakerAddress(address)._stakerAddress (contracts/Rewards.sol#104) is not in mixedCase
Parameter Rewards.setDaoAddress(address)._daoAddress (contracts/Rewards.sol#114) is not in mixedCase
Parameter Rewards.setDaoFeesPercentage(uint256)._daoFeesPercentage (contracts/Rewards.sol#124) is not in mixedCase
Event StakingstakedWithNodes(uint256) (contracts/Staking.sol#61) is not in CapWords
Event StakingwithdrawnFromNodes(uint256) (contracts/Staking.sol#63) is not in CapWords
Event StakingnodeStakingDaoFeeTransfer(address,uint256) (contracts/Staking.sol#65) is not in CapWords
Event StakingupdatedNodeStakingActiveFlag(bool) (contracts/Staking.sol#69) is not in CapWords
Parameter Staking.transferToNode(address,uint256)._nodeProxy (contracts/Staking.sol#299) is not in mixedCase
Parameter Staking.updateMinDeposit(uint256)._newMinDeposit (contracts/Staking.sol#335) is not in mixedCase
Parameter Staking.updateMaxDeposit(uint256)._newMaxDeposit (contracts/Staking.sol#341) is not in mixedCase
Parameter Staking.updateOperatorAddress(address)._operator (contracts/Staking.sol#347) is not in mixedCase
Parameter Staking.setRewardsContractAddress(Rewards)._rewardsContractAddress (contracts/Staking.sol#353) is not in mixedCase
Parameter Staking.setUndelegationContractAddress(address)._undelegationContractAddress (contracts/Staking.sol#363) is not in mixedCase
Parameter Timelock.setTimeLockOwner(address)._timelockOwner (contracts/Timelock.sol#126) is not in mixedCase
Parameter Timelock.setLockedPeriod(uint256)._lockedPeriod (contracts/Timelock.sol#136) is not in mixedCase
Parameter Undelegation.setStakingContractAddress(address)._stakingContractAddress (contracts/Undelegation.sol#75) is not in mixedCase
Parameter Undelegation.setUnbondingTime(uint256)._unbondingTime (contracts/Undelegation.sol#82) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

HederaResponseCodes.OK (contracts/HederaResponseCodes.sol#6) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_TRANSACTION (contracts/HederaResponseCodes.sol#7) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.PAYER_ACCOUNT_NOT_FOUND (contracts/HederaResponseCodes.sol#8) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_NODE_ACCOUNT (contracts/HederaResponseCodes.sol#9) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.TRANSACTION_EXPIRED (contracts/HederaResponseCodes.sol#10) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_TRANSACTION_START (contracts/HederaResponseCodes.sol#11) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_TRANSACTION_DURATION (contracts/HederaResponseCodes.sol#12) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_SIGNATURE (contracts/HederaResponseCodes.sol#13) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.MEMO_TOO_LONG (contracts/HederaResponseCodes.sol#14) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INSUFFICIENT_TX_FEE (contracts/HederaResponseCodes.sol#15) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INSUFFICIENT_PAYER_BALANCE (contracts/HederaResponseCodes.sol#16) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.DUPLICATE_TRANSACTION (contracts/HederaResponseCodes.sol#17) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.BUSY (contracts/HederaResponseCodes.sol#18) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.NOT_SUPPORTED (contracts/HederaResponseCodes.sol#19) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_FILE_ID (contracts/HederaResponseCodes.sol#21) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_ACCOUNT_ID (contracts/HederaResponseCodes.sol#22) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_CONTRACT_ID (contracts/HederaResponseCodes.sol#23) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_TRANSACTION_ID (contracts/HederaResponseCodes.sol#24) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.RECEIPT_NOT_FOUND (contracts/HederaResponseCodes.sol#25) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.RECORD_NOT_FOUND (contracts/HederaResponseCodes.sol#26) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_SOLIDITY_ID (contracts/HederaResponseCodes.sol#27) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.UNKNOWN (contracts/HederaResponseCodes.sol#29) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.SUCCESS (contracts/HederaResponseCodes.sol#30) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.FAIL_INVALID (contracts/HederaResponseCodes.sol#31) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.FAIL_FEE (contracts/HederaResponseCodes.sol#32) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.FAIL_BALANCE (contracts/HederaResponseCodes.sol#33) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.KEY_REQUIRED (contracts/HederaResponseCodes.sol#35) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.BAD_ENCODING (contracts/HederaResponseCodes.sol#36) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INSUFFICIENT_ACCOUNT_BALANCE (contracts/HederaResponseCodes.sol#37) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_SOLIDITY_ADDRESS (contracts/HederaResponseCodes.sol#38) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INSUFFICIENT_GAS (contracts/HederaResponseCodes.sol#40) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.CONTRACT_SIZE_LIMIT_EXCEEDED (contracts/HederaResponseCodes.sol#41) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.LOCAL_CALL_MODIFICATION_EXCEPTION (contracts/HederaResponseCodes.sol#42) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.CONTRACT_REVERT_EXECUTED (contracts/HederaResponseCodes.sol#43) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.CONTRACT_EXECUTION_EXCEPTION (contracts/HederaResponseCodes.sol#44) is never used in Staking (contracts/Staking.sol#20-404)
```

```
HederaResponseCodes.INVALID_RECEIVING_NODE_ACCOUNT (contracts/HederaResponseCodes.sol#45) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.MISSING_QUERY_HEADER (contracts/HederaResponseCodes.sol#46) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.ACCOUNT_UPDATE_FAILED (contracts/HederaResponseCodes.sol#48) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_KEY_ENCODING (contracts/HederaResponseCodes.sol#49) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.NULL_SOLIDITY_ADDRESS (contracts/HederaResponseCodes.sol#50) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.CONTRACT_UPDATE_FAILED (contracts/HederaResponseCodes.sol#52) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_QUERY_HEADER (contracts/HederaResponseCodes.sol#53) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_FEE_SUBMITTED (contracts/HederaResponseCodes.sol#55) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_PAYER_SIGNATURE (contracts/HederaResponseCodes.sol#56) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.KEY_NOT_PROVIDED (contracts/HederaResponseCodes.sol#58) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_EXPIRATION_TIME (contracts/HederaResponseCodes.sol#59) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.NO_WACL_KEY (contracts/HederaResponseCodes.sol#60) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.FILE_CONTENT_EMPTY (contracts/HederaResponseCodes.sol#61) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_ACCOUNT_AMOUNTS (contracts/HederaResponseCodes.sol#62) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.EMPTY_TRANSACTION_BODY (contracts/HederaResponseCodes.sol#63) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_TRANSACTION_BODY (contracts/HederaResponseCodes.sol#64) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_SIGNATURE_TYPE_MISMATCHING_KEY (contracts/HederaResponseCodes.sol#66) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_SIGNATURE_COUNT_MISMATCHING_KEY (contracts/HederaResponseCodes.sol#67) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.EMPTY_LIVE_HASH_BODY (contracts/HederaResponseCodes.sol#69) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.EMPTY_LIVE_HASH (contracts/HederaResponseCodes.sol#70) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.EMPTY_LIVE_HASH_KEYS (contracts/HederaResponseCodes.sol#71) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_LIVE_HASH_SIZE (contracts/HederaResponseCodes.sol#72) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.EMPTY_QUERY_BODY (contracts/HederaResponseCodes.sol#74) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.EMPTY_LIVE_HASH_QUERY (contracts/HederaResponseCodes.sol#75) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.LIVE_HASH_NOT_FOUND (contracts/HederaResponseCodes.sol#76) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.ACCOUNT_ID_DOES_NOT_EXIST (contracts/HederaResponseCodes.sol#77) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.LIVE_HASH_ALREADY_EXISTS (contracts/HederaResponseCodes.sol#78) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_FILE_WACL (contracts/HederaResponseCodes.sol#80) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.SERIALIZATION_FAILED (contracts/HederaResponseCodes.sol#81) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.TRANSACTION_OVERSIZE (contracts/HederaResponseCodes.sol#82) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.TRANSACTION_TOO_MANY_LAYERS (contracts/HederaResponseCodes.sol#83) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.CONTRACT_DELETED (contracts/HederaResponseCodes.sol#84) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.PLATFORM_NOT_ACTIVE (contracts/HederaResponseCodes.sol#86) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.KEY_PREFIX_MISMATCH (contracts/HederaResponseCodes.sol#87) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.PLATFORM_TRANSACTION_NOT_CREATED (contracts/HederaResponseCodes.sol#88) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_RENEWAL_PERIOD (contracts/HederaResponseCodes.sol#89) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.INVALID_PAYER_ACCOUNT_ID (contracts/HederaResponseCodes.sol#90) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.ACCOUNT_DELETED (contracts/HederaResponseCodes.sol#91) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.FILE_DELETED (contracts/HederaResponseCodes.sol#92) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.ACCOUNT_REPEATED_IN_ACCOUNT_AMOUNTS (contracts/HederaResponseCodes.sol#93) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.SETTING_NEGATIVE_ACCOUNT_BALANCE (contracts/HederaResponseCodes.sol#94) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.OBTAINER_REQUIRED (contracts/HederaResponseCodes.sol#95) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.OBTAINER_SAME_CONTRACT_ID (contracts/HederaResponseCodes.sol#96) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.OBTAINER_DOES_NOT_EXIST (contracts/HederaResponseCodes.sol#97) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.MODIFYING_IMMUTABLE_CONTRACT (contracts/HederaResponseCodes.sol#98) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.FILE_SYSTEM_EXCEPTION (contracts/HederaResponseCodes.sol#99) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.AUTORENEW_DURATION_NOT_IN_RANGE (contracts/HederaResponseCodes.sol#100) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.ERROR_DECODING_BYTESTRING (contracts/HederaResponseCodes.sol#101) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.CONTRACT_FILE_EMPTY (contracts/HederaResponseCodes.sol#102) is never used in Staking (contracts/Staking.sol#20-404)
HederaResponseCodes.CONTRACT_BYTECODE_EMPTY (contracts/HederaResponseCodes.sol#103) is never used in Staking (contracts/Staking.sol#20-404)
```

- No major issues were found by Slither.

THANK YOU FOR CHOOSING

HALBORN