Eon Delegated Staking

Horizen Labs

HALBORN

Eon Delegated Staking - Horizen Labs

Prepared by: HALBORN

Last Updated 06/24/2024

Date of Engagement by: June 3rd, 2024 - June 13th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	0	0	3	0	2

TABLE OF CONTENTS

1. Introduction

Horizen Labs engaged Halborn to conduct a security assessment on their smart contracts beginning on June 3rd, 2024, and ending on June 13th, 2024. The security assessment was scoped to the smart contracts provided in the following GitHub repository:

- HorizenLabs/eon-delegated-staking

2. Assessment Summary

The team at Halborn was provided one and a half weeks for the engagement and assigned a full-time security engineer to verify 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 assessment is to:

- Ensure that the smart contract functions operate as intended.

- Identify potential security issues within the smart contracts.

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

3. Test Approach And 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.

- Static Analysis of security for scoped contract, and imported functions. (`Slither`)

- Testnet deployment. (`Foundry`)

4. 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.

4.1 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:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 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) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

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

4.3 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.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: eon-delegated-staking

(b) Assessed Commit ID: d103171

(c) Items in scope:

- contracts/DelegatedStaking.sol
- contracts/DelegatedStakingFactory.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- 0c866b60c866b6
- b46a402b46a402
- 7fa49a37fa49a3

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CALL IS RECOMMENDED OVER TRANSFER TO SEND NATIVE ASSETS	MEDIUM	SOLVED - 06/18/2024
CHECKS-EFFECTS-INTERACTIONS PATTERN IS NOT FOLLOWED IN THE CLAIMREWARD() FUNCTION	MEDIUM	SOLVED - 06/18/2024
DELEGATEDSTAKING CONTRACT DOES NOT IMPLEMENT ANY RECEIVE OR FALLBACK FUNCTION	MEDIUM	RISK ACCEPTED
STATE VARIABLES MISSING IMMUTABLE MODIFIER	INFORMATIONAL	SOLVED - 06/18/2024
UNNECESSARY IMPORT	INFORMATIONAL	SOLVED - 06/18/2024

7. FINDINGS & TECH DETAILS

7.1 CALL IS RECOMMENDED OVER TRANSFER TO SEND NATIVE ASSETS

// MEDIUM

Description

The `DelegatedStaking` contract sends Eon tokens to the delegators by making use of the Solidity `transfer()` function:

```
function claimReward(address payable owner) external {
    (uint256 totalToClaim, ClaimData[] memory claimDetails) = calcReward(owner);
    if(totalToClaim == 0) {
        revert NothingToClaim();
    }
    //transfer reward
    owner.transfer(totalToClaim); // <----------------------
    //update last claimed epoch
    lastClaimedEpochForAddress[owner] = claimDetails[claimDetails.length - 1].epochNumber;
    emit Claim(signPublicKey, forgerVrf1, forgerVrf2, owner, claimDetails);
}
```

In Solidity, the `call()` function is often preferred over `transfer()` for sending Ether In Solidity due to some gas limit considerations:

- `transfer`: Imposes a fixed gas limit of 2300 gas. This limit can be too restrictive, especially if the receiving contract is a multisig wallet that executes more complex logic in its `receive()` function. For example, native `transfer()`calls to Gnosis Safe multisigs will always revert with an out-of-gas error in Binance Smart Chain.
- `call`: Allows specifying a custom gas limit, providing more flexibility and ensuring that the receiving contract can perform necessary operations.

It should be noted that using `call` also requires explicit reentrancy protection mechanisms (e.g., using checks-effects-interactions pattern or the `ReentrancyGuard` contract from OpenZeppelin).

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:M/R:N/S:U (6.3)

Recommendation

Consider using `call()` over `transfer()` to transfer native assets in order to ensure compatibility with any type of multisig wallet. Moreover, use the check-effects-interactions pattern in the `DelegatedStaking.claimReward()` function to avoid any reentrancy risks.

Remediation Plan

SOLVED: The **Horizen Labs team** solved the issue by implementing the recommended solution.

Remediation Hash

https://github.com/HorizenLabs/eon-delegated-staking/pull/4/commits/0c866b6492bc229827460e8c9dae54f5fb725cea

7.2 CHECKS-EFFECTS-INTERACTIONS PATTERN IS NOT FOLLOWED IN THE CLAIMREWARD() FUNCTION

// MEDIUM

Description

The `DelegatedStaking` contract implements the `claimReward()` function used by delegators to claim their rewards:

```
function claimReward(address payable owner) external {
    (uint256 totalToClaim, ClaimData[] memory claimDetails) = calcReward(owner);
    if(totalToClaim == 0) {
        revert NothingToClaim();
    }
    //transfer reward
    owner.transfer(totalToClaim); // <-----------------------------
    //update last claimed epoch
    lastClaimedEpochForAddress[owner] = claimDetails[claimDetails.length - 1].epochNumber; // <--------
-------
    emit Claim(signPublicKey, forgerVrf1, forgerVrf2, owner, claimDetails);
}
```

As we can see in the code above, the check-effects-interactions pattern is not respected as the Eon transfer(interaction) is performed before the update of the `lastClaimedEpochForAddress` state variable(effect). This could cause some reentrancy risks (which are partially mitigated by the 2300 gas limit of the `transfer()` function).

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:M/R:N/S:U (6.3)

Recommendation

It is recommended to update the `DelegatedStaking.claimReward()` function as shown below:

```
function claimReward(address payable owner) external {
    (uint256 totalToClaim, ClaimData[] memory claimDetails) = calcReward(owner);
    if(totalToClaim == 0) {
        revert NothingToClaim();
    }
    //update last claimed epoch
    lastClaimedEpochForAddress[owner] = claimDetails[claimDetails.length - 1].epochNumber; // <--------
-------
    //transfer reward
    owner.transfer(totalToClaim); // <-----------------------------
    emit Claim(signPublicKey, forgerVrf1, forgerVrf2, owner, claimDetails);
}
```

Remediation Plan

SOLVED: The **Horizen Labs team** solved the issue by implementing the recommended solution.

Remediation Hash

https://github.com/HorizenLabs/eon-delegated-staking/pull/3/commits/b46a4025b1016635f8a90d31cb9a7abd39d7b710

7.3 DELEGATEDSTAKING CONTRACT DOES NOT IMPLEMENT ANY RECEIVE OR FALLBACK FUNCTION

// MEDIUM

Description

The `DelegatedStaking` contract is supposed to hold Eon tokens in order to allow users to execute their claims by calling the `claimReward()` function. However, this contract does not implement any `payable` function, nor a `receive()` or `fallback()` function.

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation

Consider implementing a `receive()` function in the `DelegatedStaking` contract.

Remediation Plan

RISK ACCEPTED: The **Horizen Labs team** accepted the risk of this finding, and states that the smart contract will receive funds inserting its address as forger rewards receiver. In this case, fallback methods will not be required.

7.4 STATE VARIABLES MISSING IMMUTABLE MODIFIER

// INFORMATIONAL

Description

In the contract `DelegatedStaking`, the state variables `signPublicKey`, `forgerVrf1` and `forgerVrf2` can be declared as `immutable` to save some gas.
The `immutable` keyword was added to Solidity in 0.6.5. State variables can be marked `immutable` which causes them to be read-only, but only assignable in the constructor.

Score

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

Recommendation

It is recommended to add the `immutable` modifier to `signPublicKey` , `forgerVrf1` and `forgerVrf2` state variables in the `DelegatedStaking` contract.

Remediation Plan

SOLVED: The **Horizen Labs team** solved the issue by implementing the recommended solution.

Remediation Hash

https://github.com/HorizenLabs/eon-delegated-staking/pull/4/commits/7fa49a3aa2145b94957c26a465cccfffd9d523be

7.5 UNNECESSARY IMPORT

// INFORMATIONAL

Description

The `DelegatedStaking` contract declares the following import:

```
import "hardhat/console.sol";
```

`console.sol` is intended for debugging purposes and should not be part of the production code.

Score

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

Recommendation

Remove the `console.sol` import from the `DelegatedStaking` contract.

Remediation Plan

SOLVED: The **Horizen Labs team** solved the issue by implementing the recommended solution.

Remediation Hash

https://github.com/HorizenLabs/eon-delegated-staking/pull/4/commits/7fa49a3aa2145b94957c26a465cccfffd9d523be

8. AUTOMATED TESTING

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 repository 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.

DelegatedStaking.sol

```
INFO:Detectors:
DelegatedStaking.claimReward(address) (contracts/DelegatedStaking.sol#35-46) sends eth to arbitrary user
        Dangerous calls:
        - owner.transfer(totalToClaim) (contracts/DelegatedStaking.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
DelegatedStaking.calcReward(address).i (contracts/DelegatedStaking.sol#74) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
DelegatedStaking.claimReward(address).owner (contracts/DelegatedStaking.sol#35) lacks a zero-check on :
                - owner.transfer(totalToClaim) (contracts/DelegatedStaking.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Pragma version^0.8.10 (contracts/DelegatedStaking.sol#2) allows old versions
Pragma version^0.8.0 (contracts/interfaces/ForgerStakesV2.sol#2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_1 (contracts/interfaces/ForgerStakesV2.sol#55) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_2 (contracts/interfaces/ForgerStakesV2.sol#55) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_1 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_2 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_3 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_4 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_1 (contracts/interfaces/ForgerStakesV2.sol#64) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_2 (contracts/interfaces/ForgerStakesV2.sol#64) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_1 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_2 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_3 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_4 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in DelegatedStaking.claimReward(address) (contracts/DelegatedStaking.sol#35-46):
        External calls:
        - owner.transfer(totalToClaim) (contracts/DelegatedStaking.sol#41)
        State variables written after the call(s):
        - lastClaimedEpochForAddress[owner] = claimDetails[claimDetails.length - 1].epochNumber (contracts/DelegatedStaking.sol#43)
        Event emitted after the call(s):
        - Claim(signPublicKey,forgerVrf1,forgerVrf2,owner,claimDetails) (contracts/DelegatedStaking.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerablilties-4
INFO:Detectors:
DelegatedStaking.forger (contracts/DelegatedStaking.sol#12) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
DelegatedStaking.forgerVrf1 (contracts/DelegatedStaking.sol#10) should be immutable
DelegatedStaking.forgerVrf2 (contracts/DelegatedStaking.sol#11) should be immutable
DelegatedStaking.signPublicKey (contracts/DelegatedStaking.sol#9) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

DelegatedStakingFactory.sol

```
INFO:Detectors:
DelegatedStaking.claimReward(address) (contracts/DelegatedStaking.sol#35-46) sends eth to arbitrary user
        Dangerous calls:
        - owner.transfer(totalToClaim) (contracts/DelegatedStaking.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
DelegatedStaking.calcReward(address).i (contracts/DelegatedStaking.sol#74) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
DelegatedStaking.claimReward(address).owner (contracts/DelegatedStaking.sol#35) lacks a zero-check on :
                - owner.transfer(totalToClaim) (contracts/DelegatedStaking.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Pragma version^0.8.10 (contracts/DelegatedStaking.sol#2) allows old versions
Pragma version^0.8.10 (contracts/DelegatedStakingFactory.sol#2) allows old versions
Pragma version^0.8.0 (contracts/interfaces/ForgerStakesV2.sol#2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_1 (contracts/interfaces/ForgerStakesV2.sol#55) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_2 (contracts/interfaces/ForgerStakesV2.sol#55) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_1 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_2 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_3 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.registerForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_4 (contracts/interfaces/ForgerStakesV2.sol#56) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_1 (contracts/interfaces/ForgerStakesV2.sol#64) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign1_2 (contracts/interfaces/ForgerStakesV2.sol#64) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_1 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_2 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_3 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Parameter ForgerStakesV2.updateForger(bytes32,bytes32,bytes1,uint32,address,bytes32,bytes32,bytes32,bytes32,bytes32,bytes1).sign2_4 (contracts/interfaces/ForgerStakesV2.sol#65) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in DelegatedStaking.claimReward(address) (contracts/DelegatedStaking.sol#35-46):
        External calls:
        - owner.transfer(totalToClaim) (contracts/DelegatedStaking.sol#41)
        State variables written after the call(s):
        - lastClaimedEpochForAddress[owner] = claimDetails[claimDetails.length - 1].epochNumber (contracts/DelegatedStaking.sol#43)
        Event emitted after the call(s):
        - Claim(signPublicKey,forgerVrf1,forgerVrf2,owner,claimDetails) (contracts/DelegatedStaking.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
DelegatedStaking.forger (contracts/DelegatedStaking.sol#12) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
DelegatedStaking.forgerVrf1 (contracts/DelegatedStaking.sol#10) should be immutable
DelegatedStaking.forgerVrf2 (contracts/DelegatedStaking.sol#11) should be immutable
DelegatedStaking.signPublicKey (contracts/DelegatedStaking.sol#9) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

- The Ether/native assets sent to an arbitrary address issue is a false positive.
- No major issues were found by Slither.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.