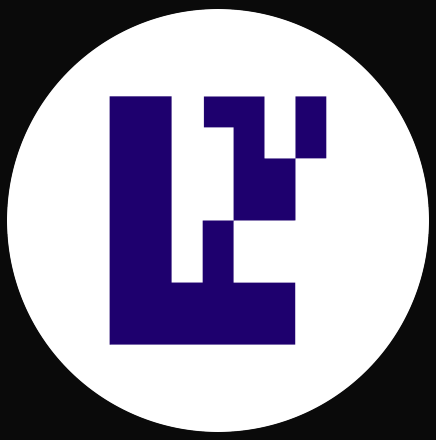
certora

Security Assessment
Draft Report



EigenLayer Moocow

June 2025

Prepared for EigenLayer team

Table of content

Project Summary **3**
Project Scope 3
Project Overview 3
Protocol Overview 4
Findings Summary 4
Severity Matrix 4
Detailed Findings **5**
Informational Issues 6
I-01. Error RefundFailed() never emitted on fee refunds 6
I-02. Correct typo for event ValidatorWithdrawn in IEigenPod.sol 8
I-03. Cache duplicate multiplication operations 9
I-04. Remove redundant source and target public key checks 10
I-05. Improvise requestConsolidation() documentation 12
Disclaimer **13**
About Certora **13**

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
EigenLayer Moocow	https://github.com/Layr-Labs/eigenlayer-contracts	48ba63b	EVM

Project Overview

This document describes the specification and verification of **EigenLayer Moocow** using manual code review findings. The work was undertaken from **June 23, 2025** to **June 27, 2025**

The following pull request is included in our scope:

- eigenlayer-contracts/pull/1425

From the PR, only changes to the following contract list are included in our scope:

```
src/contracts/interfaces/IEigenPod.sol
src/contracts/libraries/BeaconChainProofs.sol
src/contracts/pods/EigenPod.sol
src/contracts/pods/EigenPodPausingConstants.sol
src/contracts/pods/EigenPodStorage.sol
src/contracts/token/Eigen.sol
```

The team performed a manual audit of the respective Solidity contracts. During the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Protocol Overview

EigenLayer Moocow is an upgrade to the existing EigenPod implementation to support core Pectra features such as consolidation and execution layer triggerable withdrawals. By utilizing the EIP-7251 and EIP-7002 predeploys in EigenPods to ensure Pectra compatibility, this reduces the significant gas overhead involved with checkpoint processing for validators.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	-	-	-
Informational	5		
Total	5		

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
I-01	Error RefundFailed() never emitted on fee refunds	Informational	
I-02	Correct typo for event ValidatorWithdrawn in IEigenPod.sol	Informational	
I-03	Cache duplicate multiplication operations	Informational	
I-04	Remove redundant source and target public key checks	Informational	
I-05	Improvise requestConsolidation() documentation	Informational	

Informational Issues

I-01. Error RefundFailed() never emitted on fee refunds

Description: The **IEigenPod.sol** interface file introduced a new error message **RefundFailed()**. However, this is never used in EigenPod.sol which introduces redundancy in the interface.

```
None

/// @dev Thrown when refunding excess fees from a predeploy fails
error RefundFailed();
```

When consolidation requests are made using the function **requestConsolidation()**, the EigenPod owner or proof submitter is expected to overestimate the total fees supplied (msg.value) to avoid a scenario where the consolidation request fee spikes do not cause the transaction to revert. However, when the refund is processed, it uses Openzeppelin's **Address.sol** library to send the remaining value.

```
None
    function requestConsolidation(
        ConsolidationRequest[] calldata requests
    ) external payable onlyWhenNotPaused(PAUSED_CONSOLIDATIONS) onlyOwnerOrProofSubmitter {
        uint256 fee = getConsolidationRequestFee();
        require(msg.value >= fee * requests.length, InsufficientFunds());
        uint256 remainder = msg.value - (fee * requests.length);

       ... ...

       // Refund remainder of msg.value
        if (remainder > 0) {
>>            Address.sendValue(payable(msg.sender), remainder);
        }
```

This means if the refund fails, it uses the return data from the external call instead of the **RefundFailed()** error message.

```
None
function sendValue(address payable recipient, uint256 amount) internal {
        if (address(this).balance < amount) {
            revert Errors.InsufficientBalance(address(this).balance, amount);
        }

        (bool success, bytes memory returndata) = recipient.call{value: amount}("");
        if (!success) {
            _revert(returndata);
        }
    }
```

Recommendation: If the error message is intended to be used, use a low-level call directly during refunds instead of the sendValue() function. If the transaction fails, revert with the **RefundFailed()** error. If the error message is not intended to be used, consider removing it.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-02. Correct typo for event ValidatorWithdrawn in IEigenPod.sol

Description: In the **IEigenPod.sol** interface file, the natspec for the event **ValidatorWithdrawn** has a typographical error, where **validator** is misspelled as **validaor**, which reduces code readability.

```
None

/// @notice Emitted when a validaor is proven to have 0 balance at a given checkpoint
event ValidatorWithdrawn(uint64 indexed checkpointTimestamp, bytes32 indexed pubkeyHash);
```

Recommendation: We recommend correcting the typo..

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-03. Cache duplicate multiplication operations

Description: Function **requestConsolidation()** and **requestWithdrawal()** in **EigenPod.sol** calculate the total fees required by performing **fee * requests.length**. As seen in the snippet below, this calculation is performed twice, which leads to unnecessary gas consumption.

```
None
require(msg.value >= fee * requests.length, InsufficientFunds());
uint256 remainder = msg.value - (fee * requests.length);
```

Recommendation: We recommend caching the calculation into a memory variable as shown:

```
None
uint256 totalFees = fee * requests.length;
require(msg.value >= totalFees, InsufficientFunds());
uint256 remainder = msg.value - (totalFees);
```

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-04. Remove redundant source and target public key checks

Description: Function **requestConsolidation()** and **requestWithdrawal()** in **EigenPod.sol** ensure that the public keys supplied by the pod owner or proof submitter are of the correct length, as seen in the snippet below.

```
None
// requestConsolidation()
require(request.srcPubkey.length == 48, InvalidPubKeyLength());
require(request.targetPubkey.length == 48, InvalidPubKeyLength());

// requestWithdrawal()
require(request.pubkey.length == 48, InvalidPubKeyLength());
```

However, these checks are already performed in the internal function call to **_calcPubkeyHash()**.

```
None
///@notice Calculates the pubkey hash of a validator's pubkey as per SSZ spec
    function _calcPubkeyHash(
        bytes memory validatorPubkey
    ) internal pure returns (bytes32) {
        require(validatorPubkey.length == 48, InvalidPubKeyLength());
        return sha256(abi.encodePacked(validatorPubkey, bytes16(0)));
    }
```

Recommendation: In **requestConsolidation()**, we recommend removing both checks. In **requestWithdrawal()**, we recommend removing the check and moving the **_calcPubkeyHash()** internal call before the call to the predeploy to ensure public keys of invalid length are not accepted.

```
None
// requestWithdrawal()
```

```
bytes32 pubkeyHash = _calcPubkeyHash(request.pubkey);

bytes memory callData = abi.encodePacked(request.pubkey, request.amountGwei);
(bool ok,) = WITHDRAWAL_REQUEST_ADDRESS.call{value: fee}(callData);
require(ok, PredeployFailed());
```

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-05. Improvise requestConsolidation() documentation

Description: For function **requestConsolidation()** in interface **IEigenPod.sol**, one of the requirements not checked by the pod is that of source and target validators being active.

```
None
Some requirements that are NOT checked by the pod:
/// - Both the source and target validators MUST be active and MUST NOT have initiated exits
```

However, the implementation of the function **requestConsolidation()** checks for the target validator to be active. This is intentional to disallow cross-pod consolidations. However, the comment from the interface does not accurately reflect this.

```
None
require(target.status == VALIDATOR_STATUS.ACTIVE, ValidatorNotActiveInPod());
```

Recommendation: We recommend improving the comment to mention that the target validator is checked to be active to disallow cross-pod validations.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.