Eigen Labs

EigenLayer – Payment Coordinator

Security Assessment Report

Version: 1.0

May, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the EigenLayer Payment Coordinator smart contract. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the EigenLayer Payment Coordinator smart contract contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the EigenLayer Payment Coordinator smart contract.

Overview

This review targeted the upcoming Payments release for EigenLayer, which introduced a single `PaymentCoordinator` contract to enable AVSs to pay Stakers/Operators, and allow Stakers/Operators to claim payments.

Additionally, an offchain payment calculation that determines how an AVS's payment is distributed between an Operator and its Stakers was also included of this review.

The `PaymentCoordinator` does not interact with any existing M2 smart contracts. However, the offchain payment calculation ingests events and state from M2 smart contracts to determine payment distribution.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to `PaymentCoordinator.sol` file at commit 3eec97d.

Review of the offchain payment calculation docs was also conducted as a part of this assessment to understand how distribution calculations are performed offchain.

Note: third party libraries and dependencies, such as OpenZeppelin, were excluded from the scope of this assessment.

Approach

The review was conducted on the file hosted on the `eigenlayer-contracts` repository at commit 3eec97d.

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Coverage Limitations

Due to a time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 5 issues during this assessment. Categorised by their severity:

- Medium: 1 issue.

- Low: 2 issues.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the EigenLayer Payment Coordinator smart contract. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
EG4-01	Repeated Identical Range Payments Will Be Allowed In Separate Transactions	Medium	Open
EG4-02	Lack Of Access Control Checks On `initialize()`	Low	Open
EG4-03	Rounding In Calculations Leaves Tokens Unassigned	Low	Open
EG4-04	Unnecessary Same Day Registration Calculation	Informational	Open
EG4-05	Miscellaneous General Comments	Informational	Open

EG4-01	Repeated Identical Range Payments Will Be Allowed In Separate Transactions		
Asset	`PaymentCoordinator.sol`		
Status	**Open**		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

As `paymentNonce[msg.sender]` increments with every range payment and its value is used in the calculation of `rangePaymentHash` and `rangePaymentForAllHash`, all of the generated range payment hashes will be unique, regardless of their other values.

This means that the checks on line [**136**] and line [**164**] will not catch whether a range payment has been submitted before. Instead, if a range payment is submitted twice, it will be fully processed with a new and unique payment hash, which will also allow it to be successfully paid out to stakers and operators through the payment calculation system.

Recommendations

Consider not using nonces in the calculation of the payment hashes.

EG4-02	Lack Of Access Control Checks On `initialize()`		
Asset	`PaymentCoordinator.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

There are no access control checks on the `initialize()` function used to configure the contract during deployment.

If there is a period when the contracts have been deployed but not initialised, an attacker could front-run the initialisation process by calling `initialize()` to set their own parameters, such as owner address or protocol specific values.

Recommendations

Implement access controls on `initialize()` function to ensure it can only be called by specific, trusted addresses.

Ideally, ensure the entire deployment and initialisation process is performed in a single transaction.

Note, depending on the current deployment procedure, it is possible that this issue is already addressed.

EG4-03	Rounding In Calculations Leaves Tokens Unassigned		
Asset	`EigenLayer Payments Calculation`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

There are two instances in the payment calculation where rounding will leave tokens from a payment range unassigned and thus locked in the contract. Note, the number of locked tokens would be very low, so the impact of this issue is minimal.

The first instance is in `"Active Payments"` section:

```
SELECT amount/(duration/86400) as tokens_per_day
```

`duration` is always a multiple of 86400. The maximum duration multiple in the test suite is 70. Taking that as a reference point, the resultant expression `amount/70` could lose, at most, 69 tokens for each of 70 days, which is 4830 `wei` per range.

The second instance is in `"Total-Tokens"` section:

```
SELECT *,
   cast(cast(staker_proportion AS DECIMAL(38,15)) * tokens_per_day AS DECIMAL(38,0)) as total_staker_operator_payout
FROM staker_proportion
```

As explained in the document, the decimal places are truncated to 15 for the double storage format in the database. As this format is used to calculate the column `"Total Staker Operator Payouts"`, a maximum of 999 `wei` can be lost per staker per payment hash per day.

This could potentially accumulate to millions of `wei` of each payment token. In all known cases, this is still a small total value. It could only become more of a concern if a token was used with an exceptionally high unit value, such that millions of `wei` became a problem.

Recommendations

Consider calculating unassigned tokens per day and paying them back to the appropriate AVS, which could potentially be done using the existing Merkle payments system.

EG4-04	Unnecessary Same Day Registration Calculation
Asset	`EigenLayer Payments Calculation`
Status	**Open**
Rating	Informational

Description

In the section `"Operator AVS Registration Windows"`, registrations and deregistrations done on the same day are specifically filtered out by targetted queries.

However, same day registrations would all have their end date equal to or before their start time.

It would therefore be possible to remove these registrations in step 5 by changing this condition:

```
WHERE start_time != end_time
```

to:

```
WHERE start_time < end_time
```

Recommendations

Consider implementing the suggested simplification.

EG4-05	Miscellaneous General Comments
Asset	`PaymentCoordinator.sol`
Status	**Open**
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Constant names should be in block capitals**

   The constant `beaconChainETHStrategy` is not in block capitals.

2. **Gas: Check against constant before making an external call**

   On line [**341**], there is an `if` check that will pass in two circumstances. The more gas heavy check is performed first. Consider checking against the constant value first.

3. **Gas: Loops can be expensive, especially if a late iteration reverts**

   Both `payForRange()` and `payAllForRange()` loop through multiple ranges, performing all operations on each range within the loop. This could have disadvantages for long input arrays, especially if one of the last entries reverts. This is because the entire transaction would revert for all submitted ranges, and all gas paid to process those earlier ranges would be wasted.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 3 tests for test/PaymentCoordinatorUnitSigP.t.sol:PaymentCoordinatorUnitTestsSigP
[FAIL. Reason: call did not revert as expected] testFuzz_Revert_WhenDuplicateRanges_vuln(address,uint256,uint256,uint256)
[PASS] test_submitRoot_Revert_WhenForFuturePeriod() (gas: 25350)
[PASS] test_submitRoot_Revert_WhenForOlderPeriod() (gas: 101695)
Suite result: FAILED. 2 passed; 1 failed; 0 skipped; finished in 177.20ms (171.80ms CPU time)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

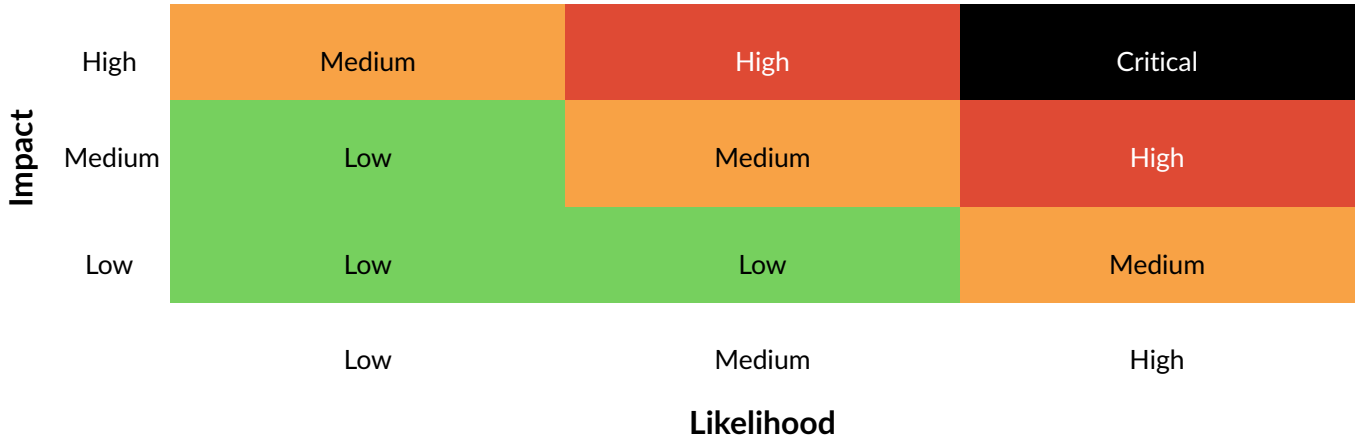

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].