Eigen Labs

EigenLayer - EIGEN Rewards

Security Assessment Report

Version: 2.0

September, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Eigen Labs smart contracts in scope. 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 Eigen Labs smart contracts 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 Eigen Labs smart contracts.

Overview

EigenLayer is a restaking protocol on Ethereum, designed to enable stakers to contribute to the cryptoeconomic security of various protocols beyond Ethereum, in return for rewards.

This review focused on an upgrade to EigenLayer's smart contracts which allows the Eigen Foundation to distribute `EIGEN` rewards to all stakers and operators across all AVSs. It also includes changes to the `bEIGEN` and `EIGEN` tokens that allow for these tokens to be minted and burnt.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the following repositories:

- eigenlayer-contracts
- EigenHopper

The scope of this time-boxed review was strictly limited to the following files at their respective commits:

- eigenlayer-contracts@decf99c
  - `core/RewardsCoordinator.sol`
  - `core/RewardsCoordinatorStorage.sol`
  - `interfaces/IRewardsCoordinator.sol`
- eigenlayer-contracts@c5d6a9d
  - `token/BackingEigen.sol`
  - `token/Eigen.sol`
- EigenHopper@2a226d6
  - `src/*`

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

Approach

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`
- Aderyn: `https://github.com/Cyfrin/aderyn`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the 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 6 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Low: 1 issue.
- Informational: 4 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Eigen Labs smart contracts in scope. 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
EGN7-01	First Submission Can Be Triggered Multiple Times	High	Resolved
EGN7-02	`bEIGEN` Cannot Be Burnt Due To Transfer Restrictions	Low	Closed
EGN7-03	Incorrect `newCooldownHorizon` Calculation	Informational	Resolved
EGN7-04	Double-Counted Submission At Cutoff	Informational	Resolved
EGN7-05	Inactive `EIGEN` Supply Stuck After Token Upgrade	Informational	Closed
EGN7-06	Miscellaneous General Comments	Informational	Closed

EGN7-01	First Submission Can Be Triggered Multiple Times		
Asset	RewardAllStakersActionGenerator.sol, TokenHopper.sol		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The first submission multiplier can be applied multiple times if `TokenHopper.pressButton()` is called repeatedly before reaching the `firstSubmissionTriggerCutoff`.

This occurs because the `generateHopperActions()` is view-only and does not maintain state of whether the initial submission has been made. Instead, it relies on the `firstSubmissionTriggerCutoff` to determine if the current submission should be treated as the first one:

```
if (block.timestamp <= firstSubmissionTriggerCutoff) {
    // ...
}
```

`TokenHopper::pressButton()` is called on a cooldown that is set during deployment in the `HopperConfiguration` struct. Depending on the time of deployment, the configured cooldown, and the `firstSubmissionTriggerCutoff`, it may be possible to submit rewards multiple times before `firstSubmissionTriggerCutoff`, allowing for these subsequent submissions to distribute more `EIGEN` tokens than intended due to the `multiple` calculation:

```
uint32 multiple = (uint32(block.timestamp) - firstSubmissionStartTimestamp) / CALCULATION_INTERVAL_SECONDS;
duration = CALCULATION_INTERVAL_SECONDS * multiple;

startTimestamp = firstSubmissionStartTimestamp;

amountsToUse[0] = amounts[0] * multiple;
amountsToUse[1] = amounts[1] * multiple;
```

Recommendations

Ensure that the configurations and deployment times of both `TokenHopper` and `RewardAllStakersActionGenerator` are correct such that `TokenHopper::pressButton()` can only be called once before and up to `firstSubmissionTriggerCutoff`.

An alternative fix involves changing the behaviour of the `generateHopperActions()` function such that it takes in a boolean `isFirstSubmission` parameter instead of relying on a timestamp cutoff. The `TokenHopper` would also need to keep track of whether `pressButton()` has been previously called to generate the correct hopper actions.

Resolution

The EigenLayer team has ensured that the first submission can only be triggered once by requiring in the deployment script that the `TokenHopper` start time is at most 1 week before the `firstSubmissionTriggerCutoff`.

This issue has been resolved in commit 9cf6f41.

EGN7-02	`bEIGEN` Cannot Be Burnt Due To Transfer Restrictions	
Asset	`BackingEigen.sol`	
Status	**Closed:** See Resolution	
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

The `burn()` function cannot be called by non-whitelisted accounts if transfer restrictions are enabled, preventing users from burning their `bEIGEN` tokens.

The `burn()` function calls `_beforeTokenTransfer()`, which has the following check:

```
// if transfer restrictions are enabled
if (block.timestamp <= transferRestrictionsDisabledAfter) {
    // if both from and to are not whitelisted
    require(
        allowedFrom[from] || allowedTo[to] || from == address(0),
        "BackingEigen._beforeTokenTransfer: from or to must be whitelisted"
    );
}
```

Since the user is not in the `allowedFrom` mapping, burning `bEIGEN` tokens is disabled as long as transfer restrictions are enabled.

Recommendations

If this behaviour is intentional, consider adding documentation and Natspec comments to the `burn()` function to clarify this behaviour.

If this behaviour is not intentional, consider adding another condition for `to == address(0)` in the `require()` function to allow users to burn their `bEIGEN` tokens:

```
// if transfer restrictions are enabled
if (block.timestamp <= transferRestrictionsDisabledAfter) {
    // if both from and to are not whitelisted
    require(
        allowedFrom[from] || allowedTo[to] || from == address(0) || to == address(0),
        "BackingEigen._beforeTokenTransfer: from or to must be whitelisted"
    );
}
```

Resolution

The EigenLayer team has acknowledged this issue with the following comment:

> *We are fine with this behaviour as transfer restrictions will be lifted before the token upgrade is completed.*

EGN7-03	Incorrect `newCooldownHorizon` Calculation
Asset	`TokenHopper.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The calculation for `newCooldownHorizon` in the `pressButton()` function does not add back `configuration.startTime` after calculating the number of seconds since the start time, resulting in a `newCooldownHorizon` value that is too small.

```
uint256 newCooldownHorizon =
    ((block.timestamp - configuration.startTime) / configuration.cooldownSeconds + 1) * configuration.cooldownSeconds;
```

Recommendations

Fix the calculation to add back `configuration.startTime` after calculating the number of seconds since the start time.

```
uint256 newCooldownHorizon =
    ((block.timestamp - configuration.startTime) / configuration.cooldownSeconds + 1) * configuration.cooldownSeconds +
      ↪  configuration.startTime;
```

Resolution

The EigenLayer team has implemented the recommended fix in commit 3084969.

EGN7-04	Double-Counted Submission At Cutoff
Asset	`RewardAllStakersActionGenerator.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

If the first submission is triggered at `firstSubmissionTriggerCutoff`, then the rewards for the period ending at `firstSubmissionTriggerCutoff` can be double-counted.

The first submission can be triggered at `firstSubmissionTriggerCutoff` due to the equality in the following check:

```
if (block.timestamp <= firstSubmissionTriggerCutoff) {
    // ...
}
```

This will account for the rewards for the periods from `[firstSubmissionStartTimestamp, firstSubmissionTriggerCutoff)` (ending at `firstSubmissionTriggerCutoff - 1`).

However, after the first submission, all subsequent submissions will count rewards only for the previous period:

```
duration = CALCULATION_INTERVAL_SECONDS;

// find the correct startTimestamp.
// RewardsSubmissions must start at a multiple of CALCULATION_INTERVAL_SECONDS
uint32 calculationIntervalNumber = uint32(block.timestamp) / CALCULATION_INTERVAL_SECONDS;
// after rounding to the latest completed calculation interval to find the end, we subtract out the duration to get the start
startTimestamp = (calculationIntervalNumber * CALCULATION_INTERVAL_SECONDS) - duration;
```

This means that if `generateHopperActions()` is called at `firstSubmissionTriggerCutoff` and then at `firstSubmissionTriggerCutoff + 1`, the rewards for the period ending at `[firstSubmissionTriggerCutoff - CALCULATION_INTERVAL_SECONDS, firstSubmissionTriggerCutoff)` will be double-counted.

This issue has an informational rating, as the `TokenHopper` will be deployed with a cooldown period of 1 week, which is the same as `CALCULATION_INTERVAL_SECONDS`. This makes it impossible for `TokenHopper::pressButton()` to be called more than once in the same period and hence the double counted rewards cannot be exploited.

Recommendations

Remove the equality from the `if` statement in `generateHopperActions` in `RewardAllStakersActionGenerator.sol`:

```
if (block.timestamp < firstSubmissionTriggerCutoff) {
    // ...
}
```

An alternative solution is to make sure the `firstSubmissionTriggerCutoff` is at the end of a period, instead of being at the start of a new period. This can be achieved by subtracting 1 from the current `firstSubmissionTriggerCutoff` value.

Resolution

The EigenLayer team has removed the equality from the `firstSubmissionTriggerCutoff` check as recommended above.

This issue has been resolved in commit a554e28.

EGN7-05	Inactive `EIGEN` Supply Stuck After Token Upgrade
Asset	`Eigen.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Any unwrapped `EIGEN` tokens before the upgrade will be stuck in the `Eigen` contract and excluded from the total supply.

In the current `EIGEN` implementation before the upgrade, unwrapping `EIGEN` sends the tokens to the `Eigen` contract.

```
function unwrap(uint256 amount) external {
    _transfer(msg.sender, address(this), amount);
    require(bEIGEN.transfer(msg.sender, amount), "Eigen.unwrap: bEIGEN transfer failed");
}
```

Any `EIGEN` tokens that have been unwrapped and locked in the `Eigen` contract will be stuck in the contract after the upgrade, as the new `EIGEN` contract now mints and burns `EIGEN` tokens upon wrapping and unwrapping.

This allows the *true* total supply of `EIGEN` to be greater than the `bEIGEN` total supply, as the `bEIGEN` can be wrapped after the upgrade to mint more `EIGEN` tokens.

This issue has an informational severity as the stuck `EIGEN` tokens cannot be used or transferred, hence they are effectively excluded from the circulating supply. This allows the circulating supply of `EIGEN` to still be 1:1 backed by `bEIGEN` that is locked in the `Eigen` contract.

Recommendations

Though the stuck `EIGEN` tokens do not affect the protocol's functionality, this discrepancy can cause confusion to integrators and external data providers that track the balances and total supply of `EIGEN`.

Consider burning the `EIGEN` tokens that are stuck in the `Eigen` contract after the upgrade.

Resolution

The EigenLayer team has acknowledged this issue with the following comment:

> *We are fine with this behaviour and do not see any real consequences from it occurring.*

EGN7-06	Miscellaneous General Comments
Asset	All contracts
Status	**Closed:** See Resolution
Rating	Informational

Description

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

1. **Missing Timestamp Validation**

   Related Asset(s): TokenHopper.sol, RewardAllStakersActionGenerator.sol

   There are multiple instances in the codebase where calculating intervals can revert due to underflow as the start timestamps are in the future.

   (a) If `RewardAllStakersActionGenerator::generateHopperActions()` is called before `firstSubmissionStartTimestamp`, the following calculation will revert due to underflow:

   ```
   uint32 multiple = (uint32(block.timestamp) - firstSubmissionStartTimestamp) / CALCULATION_INTERVAL_SECONDS;
   ```

   (b) If `TokenHopper::pressButton()` is called before `configuration.startTime`, the following calculation will revert due to underflow:

   ```
   uint256 currentPeriodStart =
       ((block.timestamp - configuration.startTime) / configuration.cooldownSeconds) * configuration.cooldownSeconds
       + configuration.startTime;
   ```

   Check at the beginning of the functions for the conditions that would cause underflow and revert gracefully using a `require()` statement for each case.

2. **Separate Conditions To Improve Code Readability**

   Related Asset(s): TokenHopper.sol, RewardAllStakersActionGenerator.sol

   There are multiple instances where the code can be broken up into smaller operations across multiple lines to improve readability and maintainability.

   (a) In `TokenHopper::_canPress()`:

   ```
   return (configuration.doesExpire ? block.timestamp < configuration.expirationTimestamp : true) &&
       (latestPress < currentPeriodStart);
   ```

   can be replaced with:

   ```
   bool isNotExpired = configuration.doesExpire ? block.timestamp < configuration.expirationTimestamp : true;
   bool isNotPressedInCurrentPeriod = latestPress < currentPeriodStart;
   return isNotExpired && isNotPressedInCurrentPeriod;
   ```

   (b) In `TokenHopper::pressButton()`:

   ```
   uint256 newCooldownHorizon =
           ((block.timestamp - configuration.startTime) / configuration.cooldownSeconds + 1) *
       ↪  configuration.cooldownSeconds;
   ```

   can be replaced with:

   ```
   uint256 currentPeriodNum = (block.timestamp - configuration.startTime) / configuration.cooldownSeconds;
   uint256 nextPeriodNum = currentPeriodNum + 1;
   uint256 nextPeriodStart = (nextPeriodNum * configuration.cooldownSeconds) + configuration.startTime;
   uint256 newCooldownHorizon = nextPeriodStart;
   ```

(c) In `RewardAllStakersActionGenerator::generateHopperActions()`:

```
// RewardsSubmissions must start at a multiple of CALCULATION_INTERVAL_SECONDS
uint32 calculationIntervalNumber = uint32(block.timestamp) / CALCULATION_INTERVAL_SECONDS;
// after rounding to the latest completed calculation interval to find the end, we subtract out the duration to get the
    ↪ start
startTimestamp = (calculationIntervalNumber * CALCULATION_INTERVAL_SECONDS) - duration;
```

can be replaced with:

```
// RewardsSubmissions must start at a multiple of CALCULATION_INTERVAL_SECONDS
uint32 calculationIntervalNumber = uint32(block.timestamp) / CALCULATION_INTERVAL_SECONDS;
// after rounding to the latest completed calculation interval to find the end, we subtract out the duration to get the
    ↪ start
uint32 latestCalculationIntervalEndTimestamp = calculationIntervalNumber * CALCULATION_INTERVAL_SECONDS;
startTimestamp = latestCalculationIntervalEndTimestamp - duration;
```

Split the calculations into smaller operations to improve readability and maintainability as shown above.

Recommendations

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

Resolution

The EigenLayer team has implemented fixes for *(1)* in commit a839ab6.

A partial fix for *(2)* has been implemented in commit 8709beb.

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 1 test for test/tests-fork/EigenUpgrade.fork.t.sol:EigenUpgradeForkTest
[SKIP] test_actualTotalSupply_MintAfterUpgrade_Vuln() (gas: 0)
Suite result: ok. 0 passed; 0 failed; 1 skipped; finished in 45.41ms (71.73µs CPU time)

Ran 1 test for test/tests-fork/BackingEigen.fork.t.sol:BackingEigenForkTest
[SKIP] test_burn_WithTransferRestrictionsEnabled_Vuln() (gas: 0)
Suite result: ok. 0 passed; 0 failed; 1 skipped; finished in 47.82ms (49.15µs CPU time)

Ran 2 tests for test/tests-fork/Eigen.fork.t.sol:EigenForkTest
[PASS] test_totalSupply() (gas: 240470)
[PASS] test_wrap_unwrap() (gas: 296390)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 52.70ms (2.14ms CPU time)

Ran 3 tests for test/tests-fork/RewardAllStakersActionGenerator.fork.t.sol:RewardAllStakersActionGeneratorForkTest
[PASS] test_generateHopperActions_BeforeFirstSubmissionStartTimestamp_Vuln() (gas: 20522)
[SKIP] test_generateHopperActions_MultipleFirstSubmissions_Vuln() (gas: 0)
[PASS] test_generateHopperActions_OverlappingSubmissions_Vuln() (gas: 220635)
Suite result: ok. 2 passed; 0 failed; 1 skipped; finished in 52.63ms (3.06ms CPU time)

Ran 4 tests for test/tests-fork/TokenHopper.fork.sol:TokenHopperForkTest
[PASS] testFuzz_canPress(uint256,uint256) (runs: 1001, µ: 159522, ~: 159634)
[PASS] test_pressButton_ButtonPressedEvent_IncorrectNewCooldownHorizon_Vuln() (gas: 577463)
[PASS] test_pressButton_RevertIf_AtExpirationTimestamp() (gas: 774647)
[PASS] test_pressButton_RevertIf_InCooldown() (gas: 744376)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 505.40ms (462.63ms CPU time)

Ran 5 test suites in 507.54ms (703.96ms CPU time): 8 tests passed, 0 failed, 3 skipped (11 total tests)
```

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.

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

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].

σ'