PERIMETER

Berachain Beacon Kit #2

Defensive Fuzzing Report

Jan. 7, 2025

Prepared By:
0xScourgedev | Lead Fuzzing Specialist
0xscourgedev@perimetersec.io

Rappie | Lead Fuzzing Specialist
rappie@perimetersec.io

Table of Contents

About Perimeter

Perimeter's mission is to deliver the highest quality fuzzing services to protocols by uniting the world's foremost fuzzing specialists. We possess extensive expertise in fuzzing a diverse range of protocols, from smaller, niche protocols to some of the largest and most complex in DeFi.

In order to deliver on our mission, we have developed the most advanced scaffolding and libraries, enabling us to create highly sophisticated fuzzing suites tailored to meet the unique challenges of each protocol.

Learn more about us at perimetersec.io.

Risk Classification

The severity of security issues identified during the security review is classified according to the table below.

A. Critical findings are highly likely to be exploited with severe impact on the protocol and require immediate attention.

B. High findings are very likely to occur, easy to exploit, or difficult but highly incentivized, and should be resolved as quickly as possible.

C. Medium findings are possible in certain circumstances or when incentivized, with a moderate likelihood of occurring, and should be addressed.

D. Low findings involve rare circumstances to exploit or offer little to no incentives, though addressing them is still recommended.

E. Informational issues represent improvements that do not impact the project's overall security but are worth considering.

Severity Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Services Provided

Perimeter has successfully delivered a comprehensive suite of services that include:

- **Fuzzing Suite Development:** Design and implement a stateful fuzzing suite using Medusa. This suite will be tailor-made for the protocol and contracts in scope. The completed fuzzing suite can later be integrated into the testing suite to serve long-term security needs.
- **Findings Reporting:** We provided thorough documentation and reporting of all findings identified throughout the engagement.
- **Invariant Testing Assurance:** Guarantee that each invariant implemented will be tested no fewer than 10,000,000 instances, ensuring thorough validation and reliability.
- **Proof-of-Concept Development:** Develop a corresponding Proof-of-Concept (PoC) for each finding and assertion/property counterexample identified, to demonstrate potential vulnerabilities and their implications.
- **Comprehensive Final Report:** Create a detailed final report that will include all findings, along with their corresponding PoCs. This report will also detail the invariants tested, their run status, and the number of runs, providing a comprehensive overview of the engagement's outcomes.

Files in Scope

The engagement will be focused on the files listed below, acquired from commit [86b41a4d292028019458f921c738527cd095](86b41a4d292028019458f921c738527cd095).

```
src
├── base
│   └── IStakingRewardsErrors.sol
├── libraries
│   ├── BeaconRoots.sol
│   ├── SSZ.sol
│   └── Utils.sol
└── pol
    ├── BeaconRootsHelper.sol
    ├── BeaconVerifier.sol
    └── interfaces
        └── IPOLErrors.sol
```

The proofs were generated using the Beacon Kit Go code, which brought many files partially or fully into scope. Given the size of the codebase, specific filenames are not listed.

Files Out of Scope

Files outside the scope were not directly considered in achieving the target. However, since many of these files are utilized by those within the scope, a significant portion was indirectly covered.

Methodology

The primary goal of this engagement was to expand the randomization of the beacon state and beacon header for the proof generation and test it against the proof validation Solidity contracts.

The Solidity contracts used for proof validation were modified from the first engagement and moved to a different repository, which can be accessed here. Additionally, a minimal `BeaconVerifier` contract was developed for this engagement, as the visibility of the validation functions were modified from external to internal.

All aspects of the beacon state and beacon header were randomized. Including but not limited to the following:

- **Genesis Validator Root**: Randomized all associated state variables.
- **Latest Beacon Block Header**: Randomized all associated state variables.
- **Block Roots**: Generated a list of random length containing randomized values.
- **State Roots**: Generated a list of random length containing randomized values.
- **Eth1Data**: Randomized all associated state variables.
- **Latest Execution Payload Header**: Randomized all associated state variables.
- **Validators**: Created a list of random length with randomized values for all state variables.
- **Balances**: Generated a list of random length containing randomized values.
- **Randdao Mixes**: Generated a list of random length.
- **Withdrawal Indices**: Randomized the next withdrawal index and next withdrawal validator index.
- **Slashings**: Generated a list of random length with randomized values.

The main threats under investigation were false positive cases, false negative cases, and potential denial-of-service vulnerabilities.

Due to the use of arbitrary execution, the effectiveness of coverage-guided fuzzing was significantly limited. As a result, we employed only Medusa to maximize the number of test runs.

Invariants

We created many tests to verify the correctness of **8** invariants described in the table below. During the execution phase, these invariants were assessed for a total of **44,000,000+** calls.

Note: Due to the usage of arbitrary execution, the number of runs is significantly less than a typical invariant suite.

The table below lists all invariants that are part of this engagement.

Invariant	Description	Tested	Passed	# Runs
PROOF-01	If the zeroValidatorPubkeyGIndex is different, the proof should never be valid	✅	✅	44.3M+
PROOF-02	If the proposerIndexGIndex is different, the proof should never be valid	✅	✅	44.3M+
PROOF-03	If the proof for verifyValidatorPubkey was not modified post-generation, then the proof should always be valid	✅	✅	44.3M+
PROOF-04	If the proof for verifyProposerIndex was not modified post-generation, then the proof should always be valid	✅	✅	44.3M+
PROOF-05	If the zeroValidatorPubkeyGIndex is the same and the proof for verifyValidatorPubkey was modified post-generation, then the proof should never be valid	✅	✅	44.3M+
PROOF-06	If the proposerIndexGIndex is the same and the proof for verifyProposerIndex was modified post-generation, then the proof should never be valid	✅	✅	44.3M+
REV-01	setZeroValidatorPubkeyGIndex never reverts	✅	✅	44.3M+
REV-02	setProposerIndexGIndex never reverts	✅	✅	44.3M+

Disclaimer

All activities conducted by Perimeter in connection with this project were carried out in accordance with the terms outlined in a Statement of Work and an agreed-upon project plan, as set forth in a proposal document delivered prior to the commencement of the project.

Security assessment projects are subject to time limitations, and as such, the findings presented in this report should not be interpreted as an exhaustive or comprehensive identification of all security issues, vulnerabilities, or defects within the target codebase. Perimeter makes no representations or warranties that the target codebase is free from defects.

Furthermore, this report is not intended to be, and should not be construed as, investment advice or a recommendation to participate in any financial transactions. The content herein does not constitute endorsements or recommendations for any financial decisions, securities, or investment strategies.