Security Assessment

Items 54 & 56 Internal Report

November 2025

Prepared for ether.fi

Disclaimer

This document is intended for **internal** use only

Item	Initial Commit Hash	Final Commit Hash
Item 54	098cf09	700dc0
Item 56	92d8324	92d8324

Low Issues

L-01. The hash created for registered validator is prone to collision

Description: The new registerBeaconValidators() function in Staking manager creates a hash of the provided parameters by using abi.encodePacked like this:

```
JavaScript
bytes32 validatorCreationDataHash = keccak256(abi.encodePacked(depositData[i].publicKey,
depositData[i].signature, depositData[i].depositDataRoot,
depositData[i].ipfsHashForEncryptedValidatorKey, bidIds[i], etherFiNode));
```

The created hash is later used to make sure the validator is created with the same parameters used upon registration.

The problem here is that abi.encodePacked() has a known issue of leading to hash collisions when being used on dynamic values (bytes, arrays, etc.). This is exactly the case here where we have the publicKey & signature fields (which are bytes) concatenated together.

Since encodePacked strips the length and metadata of dynamic values leaving, this allows the following to be possible:

```
JavaScript
bytes memory pk1 = hex"aaaa";
bytes memory sig1 = hex"bbbb";

bytes32 packedHash1 = abi.encodePacked(pk1,sig1,...)

bytes memory pk2 = hex"aa";
bytes memory sig2 = hex"aabbbb";

bytes32 packedHash2 = abi.encodePacked(pk2,sig2,...)

packedHash1 == packedHash2 (this would evaluate to true)
```

Practically this means that the caller can provide two different inputs that evaluate to the same hash, which should not be allowed in a system.

Since the function is permissioned and the likelihood of the above is quite low, the severity of this issue is low, however given the easy fix and enhanced security it would be best to fix it

Recommendation: Use abi.encode instead of encodePacked, which retains the length of original data and prevents the above vector

Customer's response: Fixed in commit 700dc0

Fix Review: Fixed

L-02. The dataRoot might differ between registration and creation phase

Description: In the new version of the code of StakingManager, a new function has been added called registerBeaconValidators(), which does the following check:

```
JavaScript
bytes32 computedDataRoot = generateDepositDataRoot(depositData[i].publicKey,
depositData[i].signature, withdrawalCredentials, initialDepositAmount);
```

```
if (computedDataRoot != depositData[i].depositDataRoot) revert IncorrectBeaconRoot();
```

This check has been removed from createBeaconValidators() function with the assumption that it cannot differ from the registration phase. However one of the parameters provided to generateDepositDataRoot() can change, which is the initialDepositAmount variable. It is a constant variable and in case the team upgrades the contracts to a new implementation with a new value for it, this would change the output of generateDepositDataRoot(), preventing validator creation, since the values have changed since the registration.

However since the computedDataRoot check has been removed, the creation process would complete, instead of failing.

Recommendation: Do the computedDataRoot check inside createBeaconValidators as well to ensure consistency and ensure parameter changes would be caught

Customer's response: Fixed in commit 700dc0

Fix Review: Fixed

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.