EigenLayer

EigenLayer

Smart Contract Security Review #3

Version: 2.0

February, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the EigenLayer smart contracts. 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 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 EigenLayer 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.

Through EigenLayer, participants can restake their liquid staking tokens (LSTs) or natively staked ETH, delegating their stake to node operators. These operators then carry out validation services for decentralised protocols in need of a decentralised validator set.

Security Assessment Summary

This review was conducted on the files hosted on the eigenlayer-contracts repository and were assessed at commit 2323207.

Retesting was performed at commit b6a3a91.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities 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 all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used 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.

Findings Summary

The testing team identified a total of 7 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.
- High: 1 issue.
- Low: 3 issues.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the EigenLayer smart contracts. 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
EGN3-01	Verifying Beacon Chain Withdrawals Break After Deneb	Critical	Resolved
EGN3-02	`withdrawalDelayBlocks` Cannot Be Initialised After M2 Upgrade	High	Resolved
EGN3-03	Incorrect Use Of EIP-712	Low	Resolved
EGN3-04	Missing Call To `_disableInitializers()` In Proxy Contract Implementation	Low	Resolved
EGN3-05	Operator AVS Registrations Cannot Be Cancelled	Low	Resolved
EGN3-06	Only Withdrawer Can Complete Queued Withdrawals	Informational	Resolved
EGN3-07	Miscellaneous General Comments	Informational	Closed

EGN3-01	Verifying Beacon Chain Withdrawals Break After Deneb		
Asset	`BeaconChainProofs.sol` `EigenPod.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The `BeaconChainProofs.verifyWithdrawal()` function does not support withdrawals that occur after the Ethereum Deneb upgrade, resulting in withdrawals being stuck or maliciously fabricated inside the `EigenPod` contract.

The Deneb upgrade adds two new fields to the `ExecutionPayload` container to account for blobs, increasing the number of fields from 15 to 17, which results in a tree height increase from 4 to 5. However, the `BeaconChainProofs` library assumes a constant tree height of 4, resulting in the following:

1. Valid withdrawal proofs that occur after Deneb are incorrectly verified to be invalid - normal users cannot withdraw their ETH from the `EigenPod` contract,
2. The `BeaconChainProofs.verifyWithdrawal()` function is now susceptible to a second pre-image attack where a malicious attacker can pose an intermediate node as a leaf to successfully verify withdrawals that don't exist on the beacon chain.

Recommendations

Account for the change in tree height after Deneb. The fixed logic will need to also still account for withdrawals that have occurred before Deneb, so consider adding logic that determines if a withdrawal is pre or post-Deneb.

Resolution

Both pre and post-Deneb tree heights are now stored as constants. The tree height value used for every beacon chain withdrawal is determined by checking the timestamp of the withdrawal against the timestamp of the Deneb fork.

This issue has been addressed in PRs #395 and #416.

EGN3-02	`withdrawalDelayBlocks` Cannot Be Initialised After M2 Upgrade		
Asset	`DelegationManager.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The `withdrawalDelayBlocks` storage variable will not be able to be initialised, hence there will be no delay on completing withdrawals in `DelegationManager`.

In the M2 upgrade, the `withdrawalDelayBlocks` storage variable has been moved from `StrategyManager` to the `DelegationManager`. However, `withdrawalDelayBlocks` can only be initialised in the `initialize()` function, which cannot be called again as the `DelegationManager` contract has already been initialised.

It is worth noting that currently risk to funds on EigenLayer Beacon Chain Strategy might not be present, though depending on future integrations with an AVS, default values for `withdrawalDelayBlocks` may prevent an AVS from responding to same block withdrawals of malicious restaked operators. This would significantly increase the severity of this bug from what is reported currently.

Recommendations

Since the `DelegationManager` contract is not being used on the Ethereum mainnet, this issue can be avoided by deploying and initialising a new `DelegationManager` proxy that points to the M2 `DelegationManager` implementation. Avoid upgrading and using the current `DelegationManager` proxy on mainnet.

Keep in mind that the M2 `StrategyManager` and `EigenPodManager` implementation contracts will need to reference the new `DelegationManager` proxy address, so the `DelegationManager` should be upgraded first.

An alternative fix is to use the `reinitialize()` modifier on a function which sets `withdrawalDelayBlocks`.

Resolution

The `withdrawalDelayBlocks` variable was changed to `minWithdrawalDelayBlocks`.

An `onlyOwner` function `setMinWithdrawalDelayBlocks()` was added to update this variable after initialisation.

This issue has been addressed in PR #439.

EGN3-03	Incorrect Use Of EIP-712		
Asset	`DelegationManagerStorage.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

In `DelegationManagerStorage`, the `DelegationApproval` and `OperatorAVSRegistration` EIP-712 types are missing the `delegationApprover` and `salt` members respectively, causing the `DELEGATION_APPROVAL_TYPEHASH` and `OPERATOR_AVS_REGISTRATION_TYPEHASH` to be incorrectly defined.

This results in integration failures with wallets and other software that support EIP-712.

Recommendations

Add the required members to their respective data types.

Resolution

`OPERATOR_AVS_REGISTRATION_TYPEHASH` has been moved to the `AVSDirectoryStorage` contract.

This issue has been addressed in PR #435.

EGN3-04	Missing Call To `_disableInitializers()` In Proxy Contract Implementation		
Asset	`DelayedWithdrawalRouter.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

In `DelayedWithdrawalRouter.sol` the `_disableInitializers()` function is not called in the constructor. This poses a risk as it allows the initializer functions to be called on the implementation contract which can lead to unexpected behaviors.

Recommendations

Add `_disableInitializers()` to the constructor.

Resolution

The recommendation has been implemented in PR #436.

EGN3-05	Operator AVS Registrations Cannot Be Cancelled		
Asset	`DelegationManager.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

EigenLayer Operators are able to provide a signature with a corresponding salt and expiry to an AVS in order to link the AVS and Operator. If an operator accidentally provides a signature to the wrong AVS this signature cannot be cancelled, and is valid until `expiry < block.timestamp`.

Though currently AVS cannot directly slash operators, the state of AVS slashing is not known. However, the risk of unintentional distribution of signature and salts may increase significantly and unexpectedly if AVS obtain the ability to slash.

Recommendations

Add functionality to be able to cancel distributed signature and salts that may have been provided unintentionally.

Resolution

AVS registration functionality has been moved to the `AVSDirectory` contract.

The `cancelSalt()` function was added to allow an operator to cancel an AVS registration with a specific `salt`.

This issue has been addressed in PR #434.

EGN3-06	Only Withdrawer Can Complete Queued Withdrawals
Asset	`DelegationManager.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `completeQueuedWithdrawal()` function can only be called by the withdrawer. This can result in the withdrawal being locked and unclaimable if the withdrawer is a smart contract that cannot call this function.

Recommendations

Consider allowing the staker that queued the withdrawal to also call `completeQueuedWithdrawal()`, or use ERC-165 to ensure that the withdrawer can call `completeQueuedWithdrawal()`.

Resolution

The project team has implemented an alternate fix by restricting the withdrawer to be the same address as the staker.

The reported issue has been acknowledged with the following comment:

> *"[The implemented fix] modifies queueWithdrawal to require that withdrawer == staker. We made this change primarily because we're seeing a large uptick in phishers targeting users by tricking them into queueing withdrawals that go to the phisher."*

Changes can be seen in PR #438.

EGN3-07	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. **`_initializeSignatureCheckingUtils()` can be removed**

   Related Asset(s): `UpgradeableSignatureCheckingUtils.sol`

   The function `_initializeSignatureCheckingUtils()` is not used. Both `DelegationManager` and `StrategyManager` have their own functions calculating the domain separator.

   Consider removing the `_initializeSignatureCheckingUtils()` function.

2. **Operator cannot deregister from AVS**

   Related Asset(s): `DelegationManager.sol`

   There's currently no way for an operator to deregister from an AVS (this functionality is strictly reserved for the AVS). The operator thus has no recourse to exit from service in case reward incentives change in the AVS.

   Consider (if not already considered) to allow operators to deregister from an AVS.

3. **Changed container field numbers in Deneb**

   Related Asset(s): `BeaconChainProofs.sol`

   The following containers will have their numbers of fields changed after the Deneb upgrade:

   (a) BeaconBlockBody: 12 fields,
   (b) ExecutionPayload and ExecutionPayloadHeader: 17 fields
   (c) BeaconState: 28 fields (updated in Capella)

   Apart from ExecutionPayload and ExecutionPayloadHeader, the number of fields do not increase their respective tree heights, so there is no impact to the contract's functionality.

   Update the constants stored inside `BeaconChainProofs` for the containers above.

4. **Named returns**

   Related Asset(s): `DelegationManager.sol, StrategyBase.sol`

   Adding a return statement when the function defines a named return variable, is redundant

   (a) `DelegationManager.undelegate()`,
   (b) `StrategyBase.deposit()`

5. **NonReentrant modifier order**

   Related Asset(s): `DelegationManager.sol, EigenPodManager.sol`

   The nonReentrant modifier should occur before all other modifiers. It is a best-practice to protect against reentrancy in other modifiers.

   (a) `completeQueuedWithdrawal()`,
   (b) `completeQueuedWithdrawals()`,
   (c) `recordBeaconChainETHBalanceUpdate()`

6. **Redundant check**

   Related Asset(s): `EigenPodManager.sol`

   The validation `require(podOwner != address(0))` is redundant in `recordBeaconChainETHBalanceUpdate()` since EigenPods cannot be initialied with `address(0)` as the `podOwner`, therefore the modifier `onlyEigenPod()` will already revert to prevent this.

   Consider removing redundant check.

7. **Contract size reduction**

   Related Asset(s): `/*`

   (a) Use custom errors instead of error messages
   (b) Wrap code inside modifiers into an internal function

Recommendations

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

Resolution

Item 3. has been resolved by removing the unused constants. Updates can be seen in PR #437.

The remaining issues have been marked as won't fix and are therefore closed.

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 `brownie` framework was used to perform these tests and the output is given below.

```
Running 1 test for test/Basic.t.sol:BasicTest
[PASS] test_VariablesWork() (gas: 2202)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.56ms

Running 3 tests for test/PauserRegistry.t.sol:PauserRegistryTest
[PASS] test_VariablesWork() (gas: 2202)
[PASS] test_setIsPauser() (gas: 37604)
[PASS] test_setUnpauser() (gas: 26932)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 8.65ms

Running 3 tests for test/Integration-oeth.t.sol:Integration
[PASS] testOETHStrategyDepositProtocolLossOfFunds() (gas: 2100654)
[PASS] testOETHStrategyDepositUserLossOfFunds() (gas: 1574800)
[PASS] test_VariablesWork() (gas: 2225)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 10.95ms

Running 8 tests for test/StrategyBase.t.sol:StrategyBaseTest
[PASS] test_VariablesWork() (gas: 2225)
[PASS] test_depositStrategy() (gas: 203704)
[PASS] test_sharesToUnderlying() (gas: 187444)
[PASS] test_sharesToUnderlyingView() (gas: 187368)
[PASS] test_strategySetup() (gas: 24371)
[PASS] test_underlyingToShares() (gas: 187435)
[PASS] test_underlyingToSharesView() (gas: 187384)
[PASS] test_withdrawStrategy() (gas: 213372)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 5.37ms

Running 4 tests for test/BeaconChainProofs.t.sol:BeaconChainProofsTest
[PASS] test_hashValidatorBLSPubkey() (gas: 7773189)
[PASS] test_verifyStateRootAgainstLatestBlockRoot() (gas: 6254387)
[PASS] test_verifyValidatorFields() (gas: 10662469)
[PASS] test_verifyWithdrawal() (gas: 379027034)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 305.85ms

Running 3 tests for test/EIP1271SignatureUtils.t.sol:EIP1271SignatureUtilsTest
[PASS] test_EOASignatureVerification(bytes32) (runs: 1000, : 14909, ~: 14919)
[PASS] test_VariablesWork() (gas: 2202)
[PASS] test_contractSignatureVerification(bytes32) (runs: 1000, : 10027, ~: 10027)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 588.95ms

Running 1 test for test/EigenPodHelper.t.sol:EigenPodHelper
[PASS] test_VariablesWork() (gas: 2258)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.74ms

Running 1 test for test/Endian.t.sol:EndianTest
[PASS] testFuzz_fromLittleEndianUint64(bytes32) (runs: 1000, : 4077, ~: 4077)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 37.46ms

Running 16 tests for test/EigenPod.t.sol:EigenPodTest
[PASS] testDeployAndVerifyNewEigenPod() (gas: 16566507)
[PASS] test_VariablesWork() (gas: 2280)
[PASS] test_WithdrawAfterFullWithdrawal() (gas: 76611296)
[PASS] test_activateRestaking() (gas: 395095)
[PASS] test_helper_setup() (gas: 393343)
[PASS] test_initialize() (gas: 310774)
[PASS] test_recoverTokens(uint256,address) (runs: 1000, : 1476441, ~: 1767146)
[PASS] test_stake() (gas: 430495)
[PASS] test_verifyAndProcessWithdrawals() (gas: 35079438)
[PASS] test_verifyBalanceUpdates_balance() (gas: 26540579)
[PASS] test_verifyBalanceUpdates_balance_not_active() (gas: 10496861)
[PASS] test_verifyWithdrawalCredentials() (gas: 11199707)
[PASS] test_verifyWithdrawalCredentials_wrongWithdrawalCredentials() (gas: 10843366)
```

```
[PASS] test_withdrawBeforeRestaking(uint256) (runs: 1000, : 433643, ~: 433643)
[PASS] test_withdrawNonBeaconChainETHBalanceWei(uint256) (runs: 1000, : 618669, ~: 618669)
[PASS] test_withdrawRestakedBeaconChainETH() (gas: 41414396)
Test result: ok. 16 passed; 0 failed; 0 skipped; finished in 1.41s

Running 17 tests for test/DelegationManager.t.sol:DelegationManagerTest
[PASS] testMigrateNonExistentRoot() (gas: 56439)
[PASS] testMigrateValidRoot() (gas: 4932733)
[PASS] test_VariablesWork() (gas: 2225)
[PASS] test_delegateTo(bytes32) (runs: 1000, : 146280, ~: 146280)
[PASS] test_delegateToBySignature(bytes32) (runs: 1000, : 248698, ~: 248689)
[PASS] test_delegateTo_withApproverSignature(bytes32) (runs: 1000, : 206164, ~: 206174)
[PASS] test_deregisterFromAVS() (gas: 160124)
[PASS] test_init_value() (gas: 22830)
[PASS] test_modifyOperatorDetails(uint32) (runs: 1000, : 19882, ~: 19882)
[PASS] test_modifyOperatorMetadataURI(string) (runs: 1000, : 146631, ~: 146286)
[PASS] test_registerAsOperator(uint32) (runs: 1000, : 132901, ~: 132901)
[PASS] test_registerOperatorToAVS() (gas: 270042)
[PASS] test_register_modifyOperatorDetails(uint32,uint32) (runs: 1000, : 133391, ~: 134738)
[PASS] test_undelegate() (gas: 153299)
[PASS] test_undelegate_force() (gas: 145131)
[PASS] test_undelegate_unauthorised() (gas: 154539)
[PASS] test_updateAVSMetadataURI(address,string) (runs: 1000, : 27264, ~: 27238)
Test result: ok. 17 passed; 0 failed; 0 skipped; finished in 2.85s

Running 4 tests for test/StrategyBaseTVLLimits.t.sol:StrategyBaseTVLLimitsTest
[PASS] testFuzz_deposit_maxPerDeposit(uint256,uint256,uint256) (runs: 1000, : 2471483, ~: 2471483)
[PASS] testFuzz_deposit_maxTotalDeposits(uint256,uint256,uint256) (runs: 1000, : 2473469, ~: 2473469)
[PASS] testFuzz_setTVLLimits(uint256,uint256,uint256,uint256) (runs: 1000, : 2173721, ~: 2175413)
[PASS] test_VariablesWork() (gas: 2258)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 4.37s

Running 15 tests for test/StrategyManager.t.sol:StrategyManagerTest
[PASS] testFail_depositIntoStrategyWithSignature(uint256,uint256) (runs: 1000, : 358717, ~: 358727)
[PASS] test_DomainSeparatorDifferentChainId() (gas: 18088)
[PASS] test_VariablesWork() (gas: 2247)
[PASS] test_addShares(uint256,address) (runs: 1000, : 159844, ~: 163922)
[PASS] test_addShares_exceedMaxStakerStrategyListLength(uint256,address) (runs: 1000, : 1566802, ~: 1566802)
[PASS] test_addShares_removeShares(uint256,uint256,address) (runs: 1000, : 159928, ~: 161523)
[PASS] test_addStrategiesToDepositWhitelist(address[]) (runs: 1000, : 3239182, ~: 3257560)
[PASS] test_depositIntoStrategy(address,uint256) (runs: 1000, : 392534, ~: 392525)
[PASS] test_depositIntoStrategy_dummyToken(address,uint256) (runs: 1000, : 274416, ~: 274402)
[PASS] test_depositIntoStrategy_inflationAttack(uint256,uint256,uint256) (runs: 1000, : 699273, ~: 699277)
[PASS] test_migrateQueuedWithdrawal() (gas: 32152)
[PASS] test_removeShares(uint256,address) (runs: 1000, : 94463, ~: 94483)
[PASS] test_removeStrategiesFromDepositWhitelist() (gas: 62940)
[PASS] test_setStrategyWhitelister(address) (runs: 1000, : 35744, ~: 35744)
[PASS] test_withdrawSharesAsTokens_StrategyManager(address,uint256,uint256,address) (runs: 1000, : 423683, ~: 423690)
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 4.36s

Running 11 tests for test/Pausable.t.sol:PausableTest
[PASS] testFuzz_onlyPauser(address) (runs: 1000, : 25696, ~: 25696)
[PASS] testFuzz_onlyUnpauser(address) (runs: 1000, : 25762, ~: 25762)
[PASS] testFuzz_onlyWhenNotPaused(uint256) (runs: 1000, : 1094181, ~: 1064306)
[PASS] testFuzz_pause_noUnpauses(uint256,uint256) (runs: 1000, : 7992423, ~: 9291595)
[PASS] testFuzz_paused(uint256) (runs: 1000, : 598919, ~: 599914)
[PASS] testFuzz_unpause_noPauses(uint256,uint256) (runs: 1000, : 7586403, ~: 6051171)
[PASS] testFuzz_whenNotPaused(uint256) (runs: 1000, : 53149, ~: 53149)
[PASS] test_VariablesWork() (gas: 2225)
[PASS] test_init() (gas: 14994)
[PASS] test_pauseAll() (gas: 1174185)
[PASS] test_setPauserRegistry() (gas: 66655)
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 30.30s

Running 16 tests for test/EigenPodManager.t.sol:EigenPodManagerTest
[PASS] test_VariablesWork() (gas: 2258)
[PASS] test_addShares_removeShares(address,uint256,uint256) (runs: 1000, : 74253, ~: 75086)
[PASS] test_createPod(address) (runs: 1000, : 394114, ~: 394114)
[PASS] test_getBlockRootAtTimestamp(uint64,bytes32) (runs: 1000, : 60139, ~: 60139)
[PASS] test_recordBeaconChainETHBalanceUpdate_positive_minus(address,int256,int256) (runs: 1000, : 460891, ~: 460891)
```

```
[PASS] test_recordBeaconChainETHBalanceUpdate_positive_minus_delegated(address,int256,int256,address) (runs: 1000, : 608796, ~:
     ↪ 608796)
[PASS] test_recordBeaconChainETHBalanceUpdate_zero_change(address) (runs: 1000, : 445378, ~: 445378)
[PASS] test_recordBeaconChainETHBalanceUpdate_zero_negative(address,int256) (runs: 1000, : 465978, ~: 465978)
[PASS] test_recordBeaconChainETHBalanceUpdate_zero_positive(address,int256) (runs: 1000, : 457314, ~: 457314)
[PASS] test_setMaxPods(uint256) (runs: 1000, : 32428, ~: 32524)
[PASS] test_setMaxPods_zero() (gas: 39975)
[PASS] test_stake(address,bytes,bytes,bytes32) (runs: 1000, : 848904, ~: 848220)
[PASS] test_stake_not32Eth(address,bytes,bytes,bytes32,uint256) (runs: 1000, : 402669, ~: 403085)
[PASS] test_updateBeaconChainOracle(address) (runs: 1000, : 26812, ~: 26812)
[PASS] test_withdrawSharesAsTokens_EigenPodManager(uint256,bytes,bytes,bytes32) (runs: 1000, : 35194296, ~: 35194081)
[PASS] test_withdrawSharesAsTokens_Negative(uint256,bytes,bytes,bytes32) (runs: 1000, : 35124271, ~: 35122992)
Test result: ok. 16 passed; 0 failed; 0 skipped; finished in 37.52s

Running 4 tests for test/Merkle.t.sol:MerkleTest
[PASS] testFuzz_merkleizeSha256(bytes32[]) (runs: 500, : 3345163, ~: 5894333)
[PASS] testFuzz_verifyInclusionSha256(bytes32[]) (runs: 1000, : 5449029, ~: 9506698)
[PASS] test_merkleizeSha256_simpleTrees() (gas: 115104)
[PASS] test_verifyInclusionSha256_simpleTrees() (gas: 57124)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 58.69s

Running 9 tests for test/Integration.t.sol:Integration
[PASS] test_VariablesWork() (gas: 2225)
[PASS] test_multiple_queueStrategiesSimultaneously(bytes,bytes,bytes32) (runs: 1000, : 1112268, ~: 1112101)
[PASS] test_multiple_queueStrategiesSimultaneouslyDifferentWithdrawer(bytes,bytes,bytes32) (runs: 1000, : 1149628, ~: 1149454)
[PASS] test_multiple_withdrawReceiveAsSharesPartial(bytes,bytes,bytes32) (runs: 1000, : 5540189, ~: 5540021)
[PASS] test_multiple_withdrawStrategiesOnlyReceiveAsShares(bytes,bytes,bytes32) (runs: 1000, : 1110528, ~: 1110361)
[PASS] test_multiple_withdrawWithoutShares(bytes,bytes,bytes32) (runs: 1000, : 726399, ~: 726236)
[PASS] test_undelegateMultipleStrategies() (gas: 23150696)
[PASS] test_undelegateWithStrategies(bytes,bytes,bytes32) (runs: 1000, : 953833, ~: 953860)
[PASS] test_withdrawWithoutShares(bytes,bytes,bytes32) (runs: 1000, : 695102, ~: 694934)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 58.68s

Ran 16 test suites: 116 tests passed, 0 failed, 0 skipped (116 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].