Zellic

Prepared for
Ryan Zarick
Isaac Zhang
LayerZero Labs

Prepared by
Aaron Esau
Quentin Lemauf
Zellic

Septemeber 25, 2025

EigenLayer DVN
Smart Contract Security Assessment

Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for LayerZero Labs on September 25th, 2025. During this engagement, Zellic reviewed EigenLayer DVN's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Does the code match the specification?
- Is the code sound?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- EigenLayer's business logic
- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped EigenLayer DVN contracts, there were no security vulnerabilities discovered.

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	0
⬜ Informational	0

2. Introduction

2.1. About EigenLayer DVN

The scope includes a slashing contract, where entities post bond when queueing slashing requests, and an AVS registrar contract.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

EigenLayer DVN Contracts

Type	Solidity
Platform	EVM-compatible

Target	DVN-AVS
Repository	https://github.com/LayerZero-Labs/DVN-AVS ↗
Version	`450aa06f116a3730874f717e6f7d4b8aa4937c8d`
Programs	`contracts/src/eigenlayer/*.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of two person-days. The assessment was conducted by two consultants over the course of one calendar day.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

Pedro Moura
Engagement Manager
pedro@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Aaron Esau
Engineer
aaron@zellic.io ↗

Quentin Lemauf
Engineer
quentin@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 25, 2025	Start of primary review period
September 25, 2025	End of primary review period

3. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

3.1. Module: LayerZeroSlasher.sol

Function: `queueRequest(string memory _description)`

This function registers a request and starts the bonding period. The `APPROVAL_WINDOW` (bonding period duration), `bondToken`, and `bondAmount` are set by the owner in advance.

Inputs

- `_description`
    - **Validation**: None. This was fixed in `PR #26` ↗ before our audit period began but after the scope was set.
    - **Impact**: Stored in the request struct.
- `msg.sender`
    - **Validation**: None.
    - **Impact**: Stored as the `bonder` in the request struct.

Branches and code coverage

Intended branches

- ☑ Requests may be queued.

Function call analysis

- `queueRequest -> safeTransferFrom`
    - **External/internal?** External.
    - **Argument control?** Only the `from` address.
    - **Impact**: Transfers the bond amount from the bonder to the contract.

Function: `refundRequest(uint256 _id, string memory _response)`

After the bonding period, anyone may call this function to refund the bonder their token. The contract owner may also call the function at any time.

Inputs

- `_id`
    - **Validation**: Must have the status `Status.QUEUED`.
    - **Impact**: Ensures that only requests that are in the queued state can be refunded. The status is changed to `Status.REFUNDED` in this function.
- `_response`
    - **Validation**: None, but it may only be set by the contract owner.
    - **Impact**: The contract owner may pass a response string to provide additional context for the refund. It is simply stored in the struct and emitted in an event.
- `msg.sender`
    - **Validation**: None. However, there are different constraints on when the function may be called based on the sender.
    - **Impact**: If the sender is the contract owner, they may call the function at any time. If the sender is not the contract owner, they may only call the function after the bonding period has elapsed.

Branches and code coverage (including function calls)

Intended branches

- [x] Callable by the owner.
- [x] Callable by a nonowner after the bonding period.

Negative behavior

- [ ] Callable by a nonowner before the bonding period.

Function call analysis

- `refundRequest -> safeTransferFrom`
    - **External/internal?** External.
    - **Argument control?** None.
    - **Impact**: Transfers the token back to the bonder.

Function: `cancelRequest(uint256 _id, string memory _response)`

Only the contract owner may call this function to cancel a queued request during the approval window (before `expiry`). Cancelling penalizes the bonder by sending the bond to the owner.

Inputs

- `_id`
    - **Validation**: The `_requests` associated with `_id` must have the status `QUEUED` — also requires `block.timestamp <= request.expiry`.
    - **Impact**: Ensures only queued, nonexpired requests can be canceled. The status is changed to `Status.CANCELLED`.
- `_response`
    - **Validation**: None.
    - **Impact**: Stored in the request struct and emitted in an event for context.

Branches and code coverage

Intended branches

- ☑ Callable by the owner before expiry (within approval window).

Negative behavior

- ☑ Callable by a nonowner.
- ☑ Callable after expiry.
- ☑ Callable when the status is not `Status.QUEUED`.

Function call analysis

- `cancelRequest -> safeTransfer`
    - **External/internal?** External.
    - **Argument control?** None by the caller — `to` is the contract owner and `amount` is the stored bond.
    - **Impact**: Transfers the bond from the contract to the owner (bonder loses the bond).

Function: `fulfillRequest(uint256 _id, IStrategy[] memory _strategies, uint256[] memory _wadsToSlash, string memory _response)`

This function executes a queued slashing request within the `APPROVAL_WINDOW`. Only the contract owner may call this. On success, the request is marked as fulfilled, EigenLayer slashing is executed, redistribution is cleared, and the bond is refunded to the original bonder.

Inputs

- `_id`
    - **Validation**: The `_requests` associated with `_id` must have the status `QUEUED` — also requires `block.timestamp <= request.expiry`.
    - **Impact**: Transitions status to `Status.FULFILLED`, persists `_response`, and stores the associated `slashId`.
- `_strategies`
    - **Validation**: Must have the same length as `_wadsToSlash`.
    - **Impact**: Forwarded to EigenLayer `slashOperator` as the strategies to slash.
- `_wadsToSlash`
    - **Validation**: Must have the same length as `_strategies`.
    - **Impact**: Forwarded to EigenLayer `slashOperator` as the corresponding amounts to slash.
- `_response`
    - **Validation**: None.
    - **Impact**: Stored in the request struct and emitted in the event.

Branches and code coverage

Intended branches

- ☑ The owner can fulfill within the approval window (inclusive of the exact expiry boundary).
- ☑ Empty arrays and zero slash amounts succeed.

Negative behavior

- ☑ A nonowner cannot fulfill.
- ☑ Cannot fulfill after expiry (`block.timestamp > request.expiry`).
- ☑ Mismatched `_strategies` / `_wadsToSlash` arrays revert.

Function call analysis

- `fulfillRequest -> ALLOCATION_MANAGER.slashOperator`
    - **External/internal?** External.
    - **Argument control?** Caller (owner) controls `_strategies` and `_wadsToSlash`. The `operator`, `avs`, and `operatorSetId` are derived from `AVS_REGISTRAR` and contract constants — `description` comes from the stored request (set by the

bonder).
    - **Impact**: Executes EigenLayer slashing and returns `slashId`, which is stored on the request.
- `fulfillRequest -> STRATEGY_MANAGER.clearBurnOrRedistributableShares`
    - **External/internal?** External.
    - **Argument control?** None by the caller — `operatorSet` is derived from `AVS_REGISTRAR` and contract constants, and `slashId` is derived from the previous call `slashOperator`.
    - **Impact**: Clears burn or redistributable shares for the operator set and `slashId`.
- `fulfillRequest -> safeTransfer`
    - **External/internal?** External.
    - **Argument control?** `to` is the stored `bonder` and `amount` is the stored `bondAmount` — both controlled by the bonder when the request was queued.
    - **Impact**: Refunds the bond to the bonder upon successful fulfillment.

3.2. Module: LayerZeroAVSRegistrar.sol

Function: `registerOperator(address registeringOperator, address _avs, uint32[] calldata operatorSetIds, bytes calldata)`

This function registers the operator for the AVS once. It is callable only by `ALLOCATION_MANAGER`.

Inputs

- `registeringOperator`
    - **Validation**: Must equal `OPERATOR`.
    - **Impact**: Verifies the expected operator.
- `_avs`
    - **Validation**: Must equal `AVS`.
    - **Impact**: Verifies the expected AVS.
- `operatorSetIds`
    - **Validation**: Length `1` and `operatorSetIds[0] == OPERATOR_SET_ID`.
    - **Impact**: Verifies the expected operator set.

Branches and code coverage

Intended branches

- ☑ Operator may be registered once.

4. Assessment Results

During our assessment on the scoped EigenLayer DVN contracts, there were no security vulnerabilities discovered.

4.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.