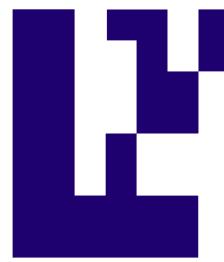




# Security Assessment

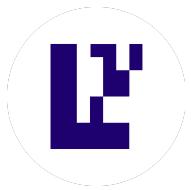
## Draft Report



# Eigenlayer - Slashing UX Improvements Audit

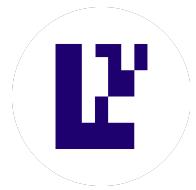
December 2025

Prepared for Eigenlayer



## Table of content

<b>Project Summary.....</b>	<b>4</b>
Project Scope.....	4
Project Overview.....	4
Threat Model.....	5
Assets.....	5
Actors.....	5
Trust Assumptions.....	5
Attack Vectors.....	6
Conclusion.....	7
Protocol Overview.....	7
Findings Summary.....	8
Severity Matrix.....	8
<b>(PR-1643) Allocation Manager Space Savings.....</b>	<b>9</b>
PR Overview.....	9
Detailed Findings.....	9
Informational Issues.....	10
I-01. getSlasher is implemented twice.....	10
<b>(PR-1645, PR-544) Slashing Commitments.....</b>	<b>11</b>
PR Overview.....	11
Detailed Findings.....	11
Medium Severity Issues.....	13
M-01 MigrateSlashers may assign an incompatible slasher address.....	13
Low Severity Issues.....	15
L-01 Instant slasher setting leaves stale slasher field in storage.....	15
Informational Issues.....	17
I-01. Re-proposing the same pending slasher restarts the delay.....	17
I-02. getPendingSlasher and getSlasher do not validate if the operator set exists.....	18
I-03. ALLOCATION_CONFIGURATION_DELAY is used both when setting the allocation delay info and changing the slasher.....	19
I-04. SlashingRegistryCoordinator::QuorumCreated event omits slasher parameter.....	20
I-05. Slasher migration can be gas-griefed by large appointee sets.....	21
<b>(PR-1655) Protocol Registry.....</b>	<b>23</b>
PR Overview.....	23
Detailed Findings.....	23
Low Severity Issues.....	24
L-01 Major version parsing truncates multi-digit majors.....	24
Informational Issues.....	26
I-01. ship lacks array length and zero-address validation.....	26
I-02. Overwriting deployment names leaves orphaned configs.....	27



I-03. configure can set configs for unshipped addresses.....	28
I-04. ship cannot change the contract name, as mentioned in the NatSpec.....	29
I-05. pauseAll can be blocked by one misconfigured deployment.....	30
<b>(PR-1502) Slashing Consolidation.....</b>	<b>31</b>
PR Overview.....	31
<b>(PR-1646) Instant Allocation for New Operators.....</b>	<b>32</b>
PR Overview.....	32
Detailed Findings.....	32
Low Severity Issues.....	33
L-01 State inconsistency for new operator allocation delay.....	33
Informational Issues.....	35
I-01. Operators may flood operator sets and disturb quorums and table updates.....	35
<b>(PR-547) Middleware.....</b>	<b>37</b>
PR Overview.....	37
Detailed Findings.....	37
Informational Issues.....	38
I-01. getNonSignerWitnessesAndApk will only work for freshly updated tables.....	38
I-02. getNonSignerWitnessesAndApk is too gas intensive.....	39
I-03. Operator info indices can be mixed up.....	40
<b>(PR-1654) Miscellaneous.....</b>	<b>41</b>
PR Overview.....	41
<b>Deployment Script Review.....</b>	<b>42</b>
1: AllocationManagerView Registration.....	42
2: StrategyBaseTVLLimits Count Source-of-Truth.....	42
3: Pauser Grant Timing Divergence Across Chains.....	42
<b>Disclaimer.....</b>	<b>43</b>
<b>About Certora.....</b>	<b>43</b>

# Project Summary

## Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
EigenLayer	<a href="#">eigenlayer-contracts</a> <a href="#">eigenlayer-middleware</a>	<a href="#">1564be7</a> <a href="#">799e94d</a>	EVM

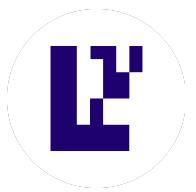
## Project Overview

This document describes the specification and verification of **EigenLayer Slashing UX Improvements PRs** using manual code review findings. The work was undertaken from **December 2nd 2025** to **December 22nd 2025**.

The following PR list is included in our scope:

[feat: slashing ux improvements #1670](#)  
[feat: ux updates #547](#)

The team performed a manual audit of all the **Solidity** contracts. During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.



## Threat Model

### Assets

- **Delegated stake (ETH / restaked tokens)** subject to slashing across operator sets
- **Slashable shares & queued-withdrawal shares** used in slashing calculations
- **Operator allocations & allocation delay state**
- **Operator set membership & quorums**
- **Slashing authority (slasher address per operator set)**
- **Protocol registry metadata**
  - Deployment addresses
  - Deployment configurations
  - Semantic version string
- **Operator tables & Merkle proofs** used for certificate verification

---

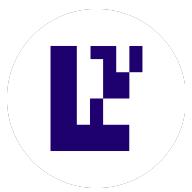
### Actors

- **Stakers / Delegators** – provide capital subject to slashing
- **Operators** – register, allocate stake, and participate in AVSs
- **AVSs (Actively Validated Services)** – define slashing rules and operator sets
- **Slasher** – authorized entity to execute slashing for an operator set
- **EigenLayer Admin / Governance**
  - DEFAULT\_ADMIN\_ROLE (ProtocolRegistry configuration)
  - PAUSER\_ROLE (emergency pause authority)
- **PermissionController (legacy)** – source of slashing appointees during migration
- **Middleware / Indexers** – rely on events and views for off-chain state reconstruction
- **Malicious or Misconfigured AVS**
- **Griefing attacker** (gas or state bloat focused)
- **Proxy admin / upgrade executor** (controls implementation + storage expectations)

---

### Trust Assumptions

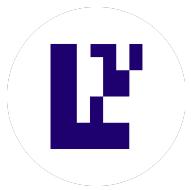
- **Slasher correctness**
  - Each operator set is assumed to have a *compatible and functional* slasher
- **Admin honesty & competence**
  - ProtocolRegistry admin is assumed not to misconfigure deployments or pause settings
- **Event completeness**
  - Off-chain systems assume emitted events fully describe on-chain state



- **Bounded data structures**
    - Operator sets, appointee sets, and tables are assumed not to grow unbounded
  - **AVS registrar defenses**
    - AVSs are assumed to handle operator registration floods safely
  - **Version parsing correctness**
    - Integrations assume semantic version helpers return accurate results
- 

## Attack Vectors

- **Slashing DoS via incompatible slasher assignment**
  - Migration can assign a slasher unable to slash certain operator sets, disabling slashing for ~17.5 days
- **Privilege confusion / authority mismatch**
  - Multiple legacy slashers collapsed into a single incompatible slasher
  - Any ways to bypass slasher delay, or invoke migration multiple times for the same operator state
- **Gas griefing**
  - Large slashing appointee sets can make `migrateSlashers()` exceed block gas limits
  - Operator-table introspection functions (`getNonSignerWitnessesAndApk()`) are prohibitively expensive on-chain
- **State inconsistency attacks**
  - Effective slasher or allocation delay exists only in “pending” state while storage shows zero values
  - Off-chain tools or integrations reading raw storage may misbehave
- **Delay manipulation**
  - Re-proposing the same pending slasher can indefinitely extend slasher-change delays
- **Censorship / emergency response failure**
  - A single misconfigured pausable deployment can block `pauseAll()`, preventing protocol-wide emergency pause
- **Operational DoS via operator flooding**
  - Instant allocation for new operators may enable rapid operator-set flooding, disturbing quorums and table updates
- **Indexing & proof mismatch**
  - Mixed operator index conventions can lead to invalid Merkle proofs and certificate verification failures
- **Monitoring & transparency gaps**
  - Missing slasher field in `QuorumCreated` events prevents accurate off-chain reconstruction
- **Version-gating failures**
  - Incorrect major version parsing can break upgrade coordination or signature-domain logic
- **Storage bloat & configuration drift**
  - Orphaned deployment configs and unshipped addresses configured in ProtocolRegistry



- **Silent misconfiguration risk**
  - Lack of input validation (array length, zero address) during deployment shipping
- **Upgrade pitfalls**
  - Consistent storage layout for all contracts that add/modify storage variables
  - AllocationManagerView storage must be aligned, so it can be called by the same proxy contract as AllocationManager
- **Overslashing**
  - The rounding during a slashing operation does not incorrectly deprive users of their shares.
  - The rounding and accounting for slashed operator shares is consistent across the whole protocol.
- **Split contract pattern pitfalls**
  - Only external view functions are casted.

---

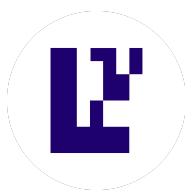
## Conclusion

Main risk is slasher/registry misconfiguration during migration/ops, which can disable slashing or block emergency pause. Secondary risk is availability griefing (gas-unbounded migrations/loops, operator flooding) that degrades AVS guarantees.

The storage consistency has been validated as well as the rounding direction for the slashing. All limitations regarding the split contract pattern have been documented and considered in the AllocationManagerView implementation.

## Protocol Overview

EigenLayer is a restaking protocol that allows operators to secure multiple AVSs by reusing delegated stake under programmable slashing and allocation rules. It provides core contracts for operator registration, delegation, allocation of slashable stake, and protocol-level safety controls. The PRs in scope are a UX-focused release that upgrades the core and middleware to improve the slashing and operator-set experience. At a high level, they move slashing authority into AllocationManager with a single slasher per operator set and a delayed update mechanism, plus a migration path for legacy operator sets. They introduce ProtocolRegistry as a global registry of deployments, semantic versioning and enable protocol-wide emergency pausing via a single call. Additional changes streamline slashing math (consolidating share-slash calculations) and improve onboarding by making the initial allocation delay effective immediately for new operators, while middleware updates align quorum creation with the new slasher model and add improved operator-table introspection.

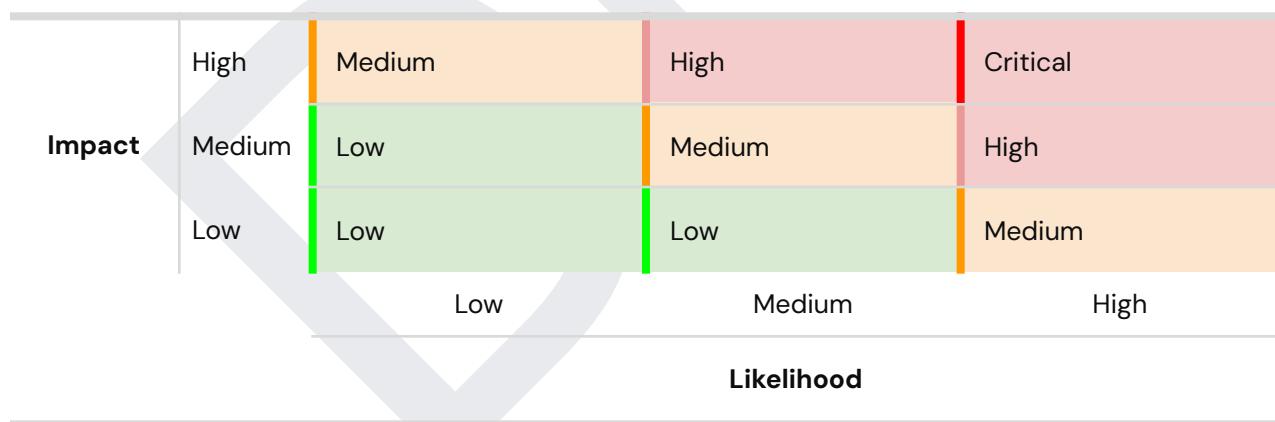


## Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	1		
Low	3		
Informational	15		
<b>Total</b>	<b>19</b>		

## Severity Matrix



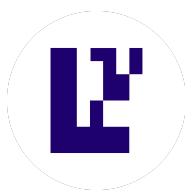
# (PR-1643) Allocation Manager Space Savings

## PR Overview

The `AllocationManager` is split into separate logic and view contracts using the `SplitContractMixin`. This split-contract pattern reduces overall contract size while cleanly separating state-changing logic from read-only functionality.

## Detailed Findings

ID	Title	Severity	Status
I-01	getSlasher is implemented twice	Informational	Pending



## Informational Issues

### I-01. getSlasher is implemented twice

**Description:** The external view methods from the `AllocationManagerView` are going to be invoked by the `AllocationManager`, using the `_delegateView` internal function.

However, `getSlasher()` is a public view function and executes the following logic directly, without delegating a call to the `AllocationManagerView`:

JavaScript

```
/// @inheritdoc IAllocationManagerView
function getSlasher(
    OperatorSet memory operatorSet
) public view returns (address) {
    SlasherParams memory params = _slashers[operatorSet.key()];

    address slasher = params.slasher;

    // If there is a pending slasher that can be applied, apply it
    if (params.effectBlock != 0 && block.number >= params.effectBlock) {
        slasher = params.pendingSlasher;
    }

    return slasher;
}
```

Furthermore, `getSlasher()` is also implemented in the `AllocationManagerView`, but it is never used by the `AllocationManager`.

**Recommendation:** Consider removing `getSlasher()` from `AllocationManagerView`.

**Customer's response:** Pending

**Fix Review:** Pending

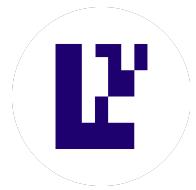
# (PR-1645, PR-544) Slashing Commitments

## PR Overview

Slashing Commitments PR moves slashing permissions from the `PermissionController` into the `AllocationManager`, so each operator set has exactly one slasher recorded on-chain. New `createOperatorSets` / `createRedistributingOperatorSets` functions accept an explicit slasher (legacy creation paths default the slasher to the AVS), and `updateSlasher` enforces the `ALLOCATION_CONFIGURATION_DELAY` (17.5 days on mainnet) for slasher changes. Existing operator sets are handled via `migrateSlashers`, which derives an initial slasher from the prior `PermissionController` configuration during upgrade. Middleware updates `SlashingRegistryCoordinator` to use the new operator-set creation flow so quorums are created with an explicit slasher.

## Detailed Findings

ID	Title	Severity	Status
M-01	MigrateSlashers may assign an incompatible slasher address	Medium	Pending
L-01	Instant slasher setting leaves stale slasher field in storage	Low	Pending
I-01	Re-proposing the same pending slasher restarts the delay	Informational	Pending
I-02	<code>getPendingSlasher</code> and <code>getSlasher</code> do not validate if the operator set exists	Informational	Pending



I-03	ALLOCATION_CONFIGURATION_DELAY is used both when setting the allocation delay info and changing the slasher	Informational	Pending
I-04	SlashingRegistryCoordinator::QuorumCreated event omits slasher parameter	Informational	Pending
I-05	Slasher migration can be gas-griefed by large appointee sets	Informational	Pending





## Medium Severity Issues

### M-01 MigrateSlashers may assign an incompatible slasher address

Severity: Medium	Impact: High	Likelihood: Low
Files:	Status:	
<a href="#">AllocationManager.sol</a>	Pending	

**Description:** The previous implementation allowed the AVS to set multiple appointees enabled for the slashing call. An AVS can have different operator sets with different strategies and many AVS implementations could have split their slasher appointees with distinct implementations for each operator set. `migrateSlashers()` can be invoked by anyone and it will automatically set the slasher of the AVS as the first appointee in the `PermissionController`:

```
JavaScript
if (slashers.length == 0 || slashers[0] == address(0)) {
    slasher = operatorSets[i].avs;
    // Else, set the slasher to the first slasher
} else {
    slasher = slashers[0];
}
```

Furthermore, there is no safe way on-chain to make sure that all AVSes have only one slasher. If there is one appointee, this does not exclude the possibility to have two slashers – the AVS and the appointee.

#### Exploit Scenario:

1. An AVS has specified two slashers as appointees:
  - a. SlasherA – responsible for slashing operatorSetA
  - b. SlasherB – responsible for slashing operatorSetB

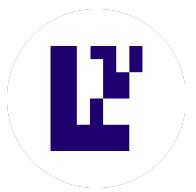
2. `migrateSlashers()` is invoked. Now slasherA is assigned as the slasher for both operator sets.
3. However, due to limitations in the implementation, it is impossible for SlasherA to slash OperatorSetB. As a result slashing will not be penalized for 17.5 days, until the slasher is updated, resulting in potential malicious behaviour.

**Recommendations:** The responsibility for slashing operator sets originates from the AVS. Due to this dependency, consider allowing only the AVS or its appointees to call `migrateSlashers()` to migrate its own operator sets (this could be done during a small window, before forceful migration).

**Customer's response:** Pending

**Fix Review:** Pending





## Low Severity Issues

### L-01 Instant slasher setting leaves stale slasher field in storage

Severity: Low	Impact: Low	Likelihood: Low
Files: <a href="#">AllocationManager.sol</a>	Status: Pending	

**Description:** When a slasher is set with an immediate effect (via `AllocationManager::createOperatorSets` or `AllocationManager::migrateSlashers` calling `_updateSlasher(..., instantEffectBlock=true)`), `_updateSlasher` only writes `pendingSlasher` and sets the `effectBlock` to `block.number`.

However, does not directly update `params.slasher` in storage, even though the new slasher is already effective. This creates an inconsistent on-chain representation where `_slashers[operatorSetKey]` can store `slasher = address(0)` while `pendingSlasher` is non-zero and `effectBlock` is the current block.

Note that `AllocationManager::getSlasher()` returns the correct slasher because it applies the pending value in-memory, therefore currently there is no serious impact.

JavaScript

```
SlasherParams memory params = _slashers[operatorSet.key()];

// If there is a pending slasher that can be applied, apply it
if (params.effectBlock != 0 && block.number >= params.effectBlock) {
    params.slasher = params.pendingSlasher;
}

// Set the pending parameters
params.pendingSlasher = slasher;
if (instantEffectBlock) {
    params.effectBlock = uint32(block.number);
} else {
    params.effectBlock = uint32(block.number) + ALLOCATION_CONFIGURATION_DELAY + 1;
}
```



**Exploit Scenario:** Right after migrating an operator set, `_slashers[key]` can be stored as `slasher=address(0)`, `pendingSlasher=0xAlice...`, `effectBlock=12345678` (current block).

**Recommendations:** When `instantEffectBlock` is used, consider updating storage so slasher reflects the effective value immediately.

**Customer's response:** Pending

**Fix Review:** Pending



## Informational Issues

### I-01. Re-proposing the same pending slasher restarts the delay

**Description:** `AllocationManager::_updateSlasher()` will always overwrite `pendingSlasher` and reset the `effectBlock`. This happens even when the AVS proposes the exact same slasher address that is already stored as `pendingSlasher`. As a result, a no-op “re-proposal” restarts the `ALLOCATION_CONFIGURATION_DELAY` countdown without changing the effective outcome. This creates unnecessary state churn and can lead to confusing timelines. Also, makes it easier to accidentally delay a planned slasher change by repeatedly submitting the same pending value.

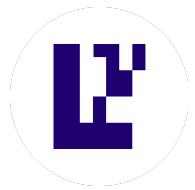
JavaScript

```
// Set the pending parameters
params.pendingSlasher = slasher;
params.effectBlock = uint32(block.number) + ALLOCATION_CONFIGURATION_DELAY +
1;
```

**Recommendation:** In `AllocationManager::_updateSlasher`, consider treating “propose the same pendingSlasher again” as a no-op and avoid resetting effectBlock. Still allow proposing `params.slasher` (the current effective slasher) to intentionally cancel/override a different pending proposal, as that is a meaningful state change.

**Customer's response:** Pending

**Fix Review:** Pending



## I-02. `getPendingSlasher` and `getSlasher` do not validate if the operator set exists

**Description:** Both `getPendingSlasher()` and `getSlasher()` do not validate if an operator set exists. As a result, if an invalid operator set is passed, it will return `address(0)` as `slasher` and `effectBlock` set to 0.

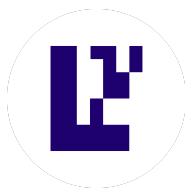
**Recommendation:** Consider validating if a slasher exists or documenting this behavior in the NatSpec.

**Customer's response:** Pending



**Fix Review:** Pending

Drop!



## I-03. ALLOCATION\_CONFIGURATION\_DELAY is used both when setting the allocation delay info and changing the slasher

**Description:** The `ALLOCATION_CONFIGURATION_DELAY` is a constant which was originally used in `_setAllocationDelay()`. However, the newly introduced `_updateSlasher` function is also using this constant, even though both actions are not related to each other:

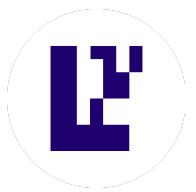
JavaScript

```
function _updateSlasher(
    OperatorSet memory operatorSet,
    address slasher,
    bool instantEffectBlock
) internal {
    ...
    // Set the pending parameters
    params.pendingSlasher = slasher;
    if (instantEffectBlock) {
        params.effectBlock = uint32(block.number);
    } else {
        params.effectBlock = uint32(block.number) + ALLOCATION_CONFIGURATION_DELAY + 1;
    }
}
```

**Recommendation:** Consider using an alternative constant for the update slasher delay.

**Customer's response:** Pending

**Fix Review:** Pending



## I-04. SlashingRegistryCoordinator::QuorumCreated event omits slasher parameter

**Description:** `SlashingRegistryCoordinator::_createQuorum` was updated to take a slasher address (used when calling `AllocationManager::createOperatorSets`), but `SlashingRegistryCoordinator::QuorumCreated` event is still emitted without this new parameter. As a result, off-chain indexers and monitoring cannot reconstruct the full quorum/operator-set configuration from event logs.

JavaScript

```
emit QuorumCreated({
    quorumNumber: quorumNumber,
    operatorSetParams: operatorSetParams,
    minimumStake: minimumStake,
    strategyParams: strategyParams,
    stakeType: stakeType,
    lookAheadPeriod: lookAheadPeriod
});
```

**Recommendation:** Consider extending `ISlashingRegistryCoordinator::QuorumCreated` event to include the slasher address and emit it from `SlashingRegistryCoordinator::_createQuorum`.

**Customer's response:** Pending

**Fix Review:** Pending



## I-05. Slasher migration can be gas-grieved by large appointee sets

**Description:** `AllocationManager::migrateSlashers` is expected to be run by the EigenLayer team to migrate legacy operator sets. For each operator set, it calls `PermissionController.getAppointees(...)`, which returns the full appointee list by copying an `EnumerableSet` to memory. OpenZeppelin explicitly notes that enumerating a full set (`EnumerableSet.values()`) is an expensive operation, with cost linearly increasing with the number of elements.

JavaScript

```
/**  
 * @dev Return the entire set in an array  
 *  
 * WARNING: This operation will copy the entire storage to memory, which can be quite  
 expensive. This is designed  
 * to mostly be used by view accessors that are queried without any gas fees. Developers  
 should keep in mind that  
 * this function has an unbounded cost, and using it as part of a state-changing function may  
 render the function  
 * uncallable if the set grows to a point where copying to memory consumes too much gas to  
 fit in a block.  
 */  
function values(Bytes32Set storage set) internal view returns (bytes32[] memory)
```

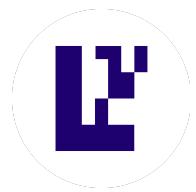
Since the AVS controls how many appointees exist for the slashing selector, an AVS can make this migration step unexpectedly expensive and potentially push the transaction over the block gas limit for its operator sets.

JavaScript

```
address[] memory slashers = permissionController.getAppointees(operatorSets[i].avs,  
address(this), this.slashOperator.selector);
```

**Recommendation:** Document that `AllocationManager::migrateSlashers` can be expensive for AVSs with many slashing appointees, since `PermissionController.getAppointees()` enumerates the full set (`EnumerableSet.values()`).

**Customer's response:** Pending



**Fix Review:** Pending

Draft

# (PR-1655) Protocol Registry

## PR Overview

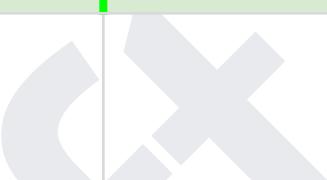
This PR introduces [ProtocolRegistry](#), a new core contract that serves as a single on-chain source of truth for protocol deployments and versioning. It maintains a registry of core proxy addresses (by name) along with per-deployment configuration, and stores a global semantic version string for the protocol. It also adds a [pauseAll](#) mechanism to pause all configured pausable deployments in a single call. As part of the upgrade, [ProtocolRegistry](#) is added as an authorized pauser in the [PauserRegistry](#) so it can perform protocol-wide emergency pausing.

## Detailed Findings

ID	Title	Severity	Status
L-01	Major version parsing truncates multi-digit majors	Low	Pending
I-01	ship lacks array length and zero-address validation	Informational	Pending
I-02	Overwriting deployment names leaves orphaned configs	Informational	Pending
I-03	configure can set configs for unshipped addresses	Informational	Pending
I-04	ship cannot change the contract name, as mentioned in the NatSpec	Informational	Pending
I-05	pauseAll can be blocked by one misconfigured deployment	Informational	Pending

## Low Severity Issues

### L-01 Major version parsing truncates multi-digit majors

Severity: Low	Impact: Low	Likelihood: Low
Files: <a href="#">ProtocolRegistry.sol</a>	Status: Pending	

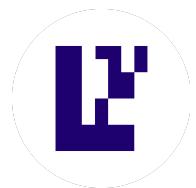
**Description:** `ProtocolRegistry::majorVersion` returns only the first byte of the stored semantic version string. This works only for single-character majors (e.g., "1.2.3" → "1"), but produces incorrect results for multi-digit majors (e.g., "10.0.0" → "1"). If the version string is prefixed (e.g., "v1.2.3"), it returns "v". This can break any on-chain or off-chain logic that relies on `ProtocolRegistry::majorVersion` for version gating, upgrade coordination, or signature/version-domain selection. It can also revert if `_semanticVersion` is the empty string, since indexing `v[0]` will panic.

JavaScript

```
/// @inheritdoc IProtocolRegistry
function majorVersion() external view returns (string memory) {
    bytes memory v = bytes(_semanticVersion.toString());
    return string(abi.encodePacked(v[0]));
}
```

**Exploit Scenario:** After a future upgrade where the protocol version becomes "10.0.0", `ProtocolRegistry::majorVersion` will still return "1". Any integration or tooling that uses `ProtocolRegistry::majorVersion` to decide which "major version" rules to apply will make the wrong decision and may break until fixed.

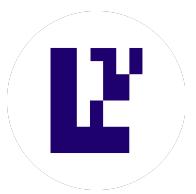
**Recommendations:** Consider parsing and returning the full major component (all digits up to the first .), and handle unexpected formats safely (e.g., optional v prefix and empty/invalid strings).



**Customer's response:** Pending

**Fix Review:** Pending

Draft



## Informational Issues

### I-01. ship lacks array length and zero-address validation

**Description:** `ProtocolRegistry::ship` takes three parallel arrays (`addresses`, `configs`, `names`) but does not validate that they have the same length before iterating over `addresses.length`. If `configs` or `names` are shorter, the call will revert “array out-of-bounds” panic. If `configs/names` are longer, the extra entries are silently ignored, increasing the risk of incomplete deployments being shipped.

JavaScript

```
for (uint256 i = 0; i < addresses.length; ++i) {
    // Append each provided
    _appendDeployment(addresses[i], configs[i], names[i]);
}
```

**Recommendation:** Consider adding input validation in `ProtocolRegistry::ship` to (1) require `addresses.length == configs.length == names.length`, and (2) reject address(0) in `addresses` (and consider rejecting empty names as well).

**Customer's response:** Pending

**Fix Review:** Pending



## I-02. Overwriting deployment names leaves orphaned configs

**Description:** In `ProtocolRegistry::ship`, when an existing name is re-shipped with a new address, the previous address keeps its `_deploymentConfigs[oldAddr]` entry, but there is no longer any name pointing to `oldAddr`. Since the registry does not expose a “get config by address” getter (and `_deploymentConfigs` is internal), the old config becomes effectively unreachable and permanently stored, creating silent storage bloat and potential operational confusion during upgrades.

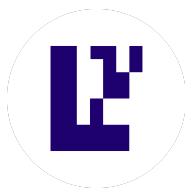
JavaScript

```
// Store name => address mapping
_deployments.set({key: _unwrap(name.toShortString()), value: addr});
// Store deployment config
_deploymentConfigs[addr] = config;
```

**Recommendation:** In `ProtocolRegistry::_appendDeployment`, if name already exists and the stored address is being overwritten, consider deleting `_deploymentConfigs[oldAddr]`. This prevents leaving behind an unreachable config for the old address and keeps the registry state clean.

**Customer's response:** Pending

**Fix Review:** Pending



## I-03. configure can set configs for unshipped addresses

**Description:** `ProtocolRegistry::configure` allows the admin to write a `DeploymentConfig` for any `addr` without checking that the address is currently registered in `_deployments` (has a name shipped via ship). This makes it easy to accidentally configure a wrong/typo address and silently create a config entry that is not tied to any deployment name. Since views (`getAddress` / `getDeployment` / `getAllDeployments`) and `pauseAll` operate over `_deployments`, these “orphan” configs are effectively junk state and can also confuse offchain monitoring because `DeploymentConfigured` is still emitted.

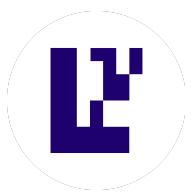
JavaScript

```
/// @inheritdoc IProtocolRegistry
function configure(
    address addr,
    DeploymentConfig calldata config
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    // Update the config
    _deploymentConfigs[addr] = config;
    // Emit the event.
    emit DeploymentConfigured(addr, config);
}
```

**Recommendation:** Consider restricting `ProtocolRegistry::configure` to only allow configuration of addresses that are already shipped/registered (ex. by configuring by name instead of raw address). This will reduce the risk of junk configs and admin misconfiguration.

**Customer's response:** Pending

**Fix Review:** Pending



## I-04. ship cannot change the contract name, as mentioned in the NatSpec

**Description:** The `ship()` function has the following comment in its NatSpec:

JavaScript

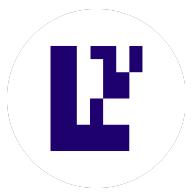
```
/// @dev Contract names can be overridden any number of times.
```

This statement is not accurate as contract names are actually used as keys, which will point to the contract address, so there is no way to change the contract name using `ship`. There is an alternative approach to call `ship` with a new name for the old address, but this will create a storage inconsistency in which there would be two names that are going to point to the same address.

**Recommendation:** Consider updating the misleading comment.

**Customer's response:** Pending

**Fix Review:** Pending



## I-05. pauseAll can be blocked by one misconfigured deployment

**Description:** `ProtocolRegistry::pauseAll` iterates over all shipped deployments and calls `IPausable(addr).pauseAll()` for those marked pausable and not deprecated. If any entry is misconfigured as pausable but does not actually implement `pauseAll()`, the entire transaction reverts and none of the deployments are paused. Given the purpose of this function is to provide a single emergency transaction to pause the protocol, allowing a single bad entry to block the whole loop creates an avoidable operational risk and can delay emergency response until the registry is corrected.

JavaScript

```
/// @inheritdoc IProtocolRegistry
function pauseAll() external onlyRole(PAUSER_ROLE) {
    uint256 length = totalDeployments();
    // Iterate over all stored deployments.
    for (uint256 i = 0; i < length; ++i) {
        (, address addr) = _deployments.at(i);
        DeploymentConfig memory config = _deploymentConfigs[addr];
        // Only attempt to pause deployments marked as pausable.
        if (config.pausable && !config.deprecated) {
            IPausable(addr).pauseAll();
        }
    }
}
```

**Recommendation:** Keeping the registry updated is of great importance in order to be able to pause using `pauseAll`. We recommend documenting this behaviour and cautiously maintaining the registry.

**Customer's response:** Pending

**Fix Review:** Pending

# (PR-1502) Slashing Consolidation

## PR Overview

This PR consolidates slashed-share accounting by deprecating `scaleForBurning` and switching `_getSlashableSharesInQueue` to use `calcSlashedAmount` instead. The change removes two parallel implementations for computing “shares to slash” and standardizes the calculation across operator shares and queued-withdrawal shares. This, also, fixes a formal verification property related to preventing overslashing and reduces the risk of divergence between “operator” and “queued” slashing logic.

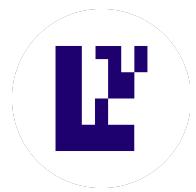
# (PR-1646) Instant Allocation for New Operators

## PR Overview

New operators can now set their initial allocation delay with immediate effect during registration, enabling them to allocate magnitude right away instead of waiting ALLOCATION\_CONFIGURATION\_DELAY (17.5 days on mainnet). The ALLOCATION\_CONFIGURATION\_DELAY still applies to later modifications of an already-set allocation delay, preserving the safety window for existing stakers. The rationale is that a newly created operator has no delegated stake, so delaying the first allocation delay provides little protection while significantly degrading onboarding UX.

## Detailed Findings

ID	Title	Severity	Status
L-01	State inconsistency for new operator allocation delay	Low	Pending
I-01	Operators may flood operator sets and disturb quorums and table updates	Informational	Pending



## Low Severity Issues

### L-01 State inconsistency for new operator allocation delay

Severity: Low	Impact: Low	Likelihood: Low
Files: <a href="#">AllocationManager.sol</a>	Status: Pending	

**Description:** When a new operator registers through `DelegationManager`, their allocation delay is set via `AllocationManager::setAllocationDelay`, which calls `_setAllocationDelay(..., newlyRegistered=true)` and sets `effectBlock = block.number`. However, `_setAllocationDelay` does not immediately update `delay/isSet`; it only updates `pendingDelay` and relies on the “apply pending if `block.number >= effectBlock`” logic. This creates a temporary storage inconsistency where `_allocationDelayInfo[operator]` can have `delay = 0` and `isSet = false`, while `pendingDelay` is non-zero and `effectBlock` is already the current block.

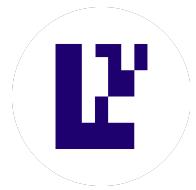
Note: `AllocationManager::getAllocationDelay` returns the correct values because it applies the pending delay in-memory, so the impact is limited to inconsistent stored fields.

JavaScript

```
function _setAllocationDelay(
    address operator,
    uint32 delay,
    bool newlyRegistered
) internal {
    // ...

    // If there is a pending delay that can be applied now, set it
    if (info.effectBlock != 0 && block.number >= info.effectBlock) {
        info.delay = info.pendingDelay;
        info.isSet = true;
    }

    info.pendingDelay = delay;
```



```
// ...  
}
```

**Exploit Scenario:** Right after an operator registers, `_allocationDelayInfo[operator]` can be stored as `delay=0` and `isSet=false`, while `pendingDelay=100` and `effectBlock=block.number`.

**Recommendations:** Consider updating the stored `AllocationDelayInfo` immediately for newly registered operators (when the delay is intended to take effect right away) so that `delay` and `isSet` reflect the effective value instead of leaving it only in `pendingDelay`.

**Customer's response:** Pending

**Fix Review:** Pending

Draft



## Informational Issues

### I-01. Operators may flood operator sets and disturb quorums and table updates

**Description:** In `modifyAllocation()`, the following comments suggest the potential behavior of the `registerOperator()` call to the `avsRegistrar`:

JavaScript

```
// Check that the operator set exists and get the operator's registration status // Operators  
do not need to be registered for an operator set in order to allocate // slashable magnitude  
to the set. In fact, it is expected that operators will // allocate magnitude before  
registering, as AVS's will likely only accept // registrations from operators that are  
already slashable.
```

This indicates that apart from protection for the stakers, the initial allocation delay that all operators have may also serve as a delay for AVS registration.

In `registerForOperatorSets()`, the following storage changes are applied:

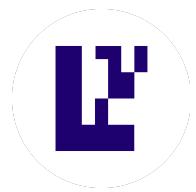
JavaScript

```
registeredSets[operator].add(operatorSet.key());  
_operatorSetMembers[operatorSet.key()].add(operator);
```

As a result, the `_operatorSetMembers` is increased. Therefore, apart from instant allocations for new operators, the change may indirectly introduce instant registration for new operators in the context of some AVS registrars (Eg: registrars that only require a non-zero allocation in order to register an operator).

Instantly registering operators, without a delay may introduce a risk for some AVS implementations of being flooded with operators. In the context of the middleware contracts, this may DOS operator table upgrades.

While in the previous version, flooding an operator set was an attack that required a long preparation and could be extinguished quickly using `deregisterFromOperatorSets()`, now this attack can be executed instantly and repeatedly using fresh new operators.



**Note:** This is a theoretical issue that is based on assumptions on potential avsRegistrar behavior. This issue highlights how a previously constrained attack vector has become operationally more feasible.

**Recommendation:** We recommend clearly communicating this potential attack vector to AVSes so that they can apply suitable defence mechanisms in case they were susceptible to this attack.

**Customer's response:** Pending

**Fix Review:** Pending

# (PR-547) Middleware

## PR Overview

This PR extends `BN254TableCalculatorBase` with on-chain introspection helpers around the operator table produced by `calculateOperatorTable`. It adds a view to construct the nonsigner witness APK by passing an explicit list of signing operators, improving verifiability and debugging without off-chain reconstruction. It also introduces a utility to return the 0-based index of a given operator within the computed operator table.

## Detailed Findings

ID	Title	Severity	Status
I-01	getNonSignerWitnessesAndApk will only work for freshly updated tables	Informational	Pending
I-02	getNonSignerWitnessesAndApk is too gas expensive	Informational	Pending
I-03	Operator info indices can be mixed up	Informational	Pending



## Informational Issues

### I-01. `getNonSignerWitnessesAndApk` will only work for freshly updated tables

**Description:** `getNonSignerWitnessesAndApk()` computes the `nonSignerWitnesses` array using the latest operator weights stored on-chain. The function is intended to help preview inputs for `BN254CertificateVerifier::verifyCertificate()`.

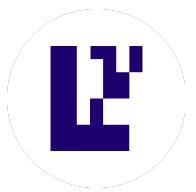
However, `verifyCertificate()` validates certificates against the operator set state at `referenceTimestamp`, not at `block.timestamp`. In most real scenarios, these two timestamps will differ. As a result, the operator weights, indexes, and Merkle proofs returned by `getNonSignerWitnessesAndApk()` will not correspond to the operator set state expected by `verifyCertificate()`.

Consequently, the output of `getNonSignerWitnessesAndApk()` is only valid when the operator table has been freshly updated and the current operator set state exactly matches the state at `referenceTimestamp`. In all other cases, the generated `nonSignerWitnesses` will be inconsistent with the verification logic and unsuitable for use in `verifyCertificate()`.

**Recommendation:** Consider documenting this limitation.

**Customer's response:** Pending

**Fix Review:** Pending



## I-02. `getNonSignerWitnessesAndApk` is too gas intensive

**Description:** `BN254TableCalculatorBase::getNonSignerWitnessesAndApk` rebuilds the registered operator table, does a nested scan to check whether each operator is in `signingOperators`, and generates a Merkle proof for every non-signer. Gas grows quickly with table size; in a benchmark with 70 registered operators and no signers (all non-signers), it used ~11.3M gas. Even though it is a view, using it from a state-changing transaction (directly or via another contract) can easily run out of gas.

JavaScript

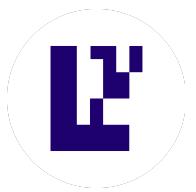
```
uint256 nonSignerCount = 0;
for (uint256 idx = 0; idx < registeredOperators.length; idx++) {
    bool signerFound = false;
    for (uint256 k = 0; k < signingOperators.length; k++) {
        // ...
    }
    // ...
}
```

**Recommendation:** Document in NatSpec and external docs that

`BN254TableCalculatorBase::getNonSignerWitnessesAndApk` is intended for off-chain `eth_call` usage and possible on-chain transactions calling it may drain callers ether balance due to gas costs.

**Customer's response:** Pending

**Fix Review:** Pending



### I-03. Operator info indices can be mixed up

**Description:** BN254TableCalculatorBase::getOperatorIndex returns the compacted 0-based index used by BN254TableCalculatorBase::calculateOperatorTable (only key-registered operators, no holes). However, BN254TableCalculatorBase::getOperatorInfos returns an array sized to the full candidate list and can contain “holes” for unregistered operators, so its array positions do not match operatorIndex. Integrators may mix these two index conventions and build invalid operatorIndex/Merkle proofs, causing certificate verification failures.

**Recommendation:** Align the functions indexing so it cannot be confused: either make getOperatorInfos return a compacted list consistent with the operator table (ideally including operator addresses), or explicitly document that getOperatorInfos is sparse and must not be used to derive operatorIndex.

**Customer's response:** Pending

**Fix Review:** Pending

# (PR-1654) Miscellaneous

## PR Overview

This PR reduces core contract bytecode size by removing `SemVerMixin` from contracts that do not use it for signature-domain separation (like those not inheriting `SignatureUtilsMixin`). It also adds a non-reverting `_canCall` helper in `PermissionControllerMixin` to save space and support “check capability” flows without forcing a revert.

# Deployment Script Review

## 1: AllocationManagerView Registration

- **Details:** `AllocationManagerView` is deployed in Script 5 but is not registered in `ProtocolRegistry`. As a core EigenLayer split-view contract, it can be registered with `{ pausable: false, deprecated: false }` since it is not Pausable. Furthermore, some additional implementations and beacons are included in the script.

**Customer's response:** Pending

**Fix Review:** Pending

## 2: StrategyBaseTVLLimits Count Source-of-Truth

- **Details:** Script 5 iterates over TVL limits using `strategyBaseTVLLimits_Count()` as the loop bound. The source-of-truth for this count is unclear. If externally inflatable, this may cause upgrade-time self-DoS via unbounded iteration.

**Customer's response:** Pending

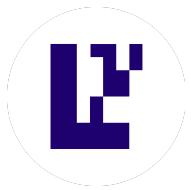
**Fix Review:** Pending

## 3: Pauser Grant Timing Divergence Across Chains

- **Details:** Pauser assignment timing differs by chain. Destination Script 5 grants `ProtocolRegistry` pauser status immediately via `setIsPauser` (outside timelock), while source/mainnet grants it via the timelocked executor bundle. In v1.9.0-slashing-ux, `setIsPauser` occurs during scheduling, whereas in v1.9.0-slashing-ux-destination it occurs immediately.

**Customer's response:** Pending

**Fix Review:** Pending



# 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.