



Security Assessment

Final Report



cap
September 2025

Prepared for cap

Table of contents

Project Summary.....	3
Project Scope.....	3
Project Overview.....	3
Protocol Overview.....	3
Findings Summary.....	4
Severity Matrix.....	4
Detailed Findings.....	5
Medium Severity Issues.....	6
M-01 Stuck rewards in first distribution due to delegation activation delay.....	6
M-02 Incorrect reward accounting causes unclaimable and stuck rewards.....	8
Low Severity Issues.....	10
L-01 Revert in reward distribution process in an edge case.....	10
L-02 External operators can register to cap's operator sets without authorization.....	12
L-03 Incorrect ERC-7201 storage location in EigenServiceManagerStorageUtils.....	14
Informational Issues.....	16
I-01. Inefficient validation order in registerStrategy().....	16
I-02. Incorrect NatSpec claims allocation can happen after one block.....	16
I-03. Misleading variable name epochDuration causes confusion between time units and epoch counts....	17
Disclaimer.....	19
About Certora.....	19

Project Summary

Project Scope

Project Name	Repository (link)	Initial Commit Hash	Platform
cap	cap-labs-dev/cap-contracts	9d692cf	EVM

Project Overview

This document describes the verification of **cap** using manual code review. The work was undertaken from **September 15th, 2025** to **September 22nd, 2025**.

The team performed a manual audit of the following Solidity contracts:

- contracts/delegation/providers/eigenlayer/EigenServiceManager.sol
- contracts/delegation/providers/eigenlayer/EigenOperator.sol
- contracts/storage/EigenServiceManagerStorageUtils.sol
- contracts/storage/EigenOperatorStorageUtils.sol

During the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Protocol Overview

cap is a capital-efficient stablecoin lending protocol that connects liquidity providers, restakers and operators in a three-party system. The protocol issues cUSD (1:1 backed stablecoin) and stcUSD (yield-bearing version) while uniquely leveraging SSN infrastructure for collateralization rather than traditional validation services.

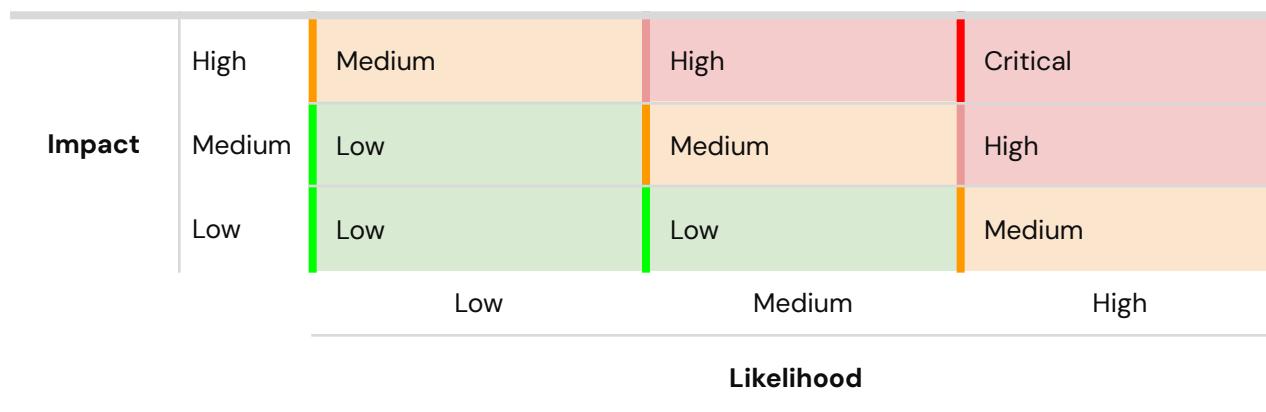
The audit focuses on cap's integration with EigenLayer as a Shared Security Network provider and the repurposing of EigenLayer's delegation and slashing mechanisms for collateralized lending and liquidations.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	0	-	-
High	0	-	-
Medium	2	-	-
Low	3	-	-
Informational	3	-	-
Total	8	-	-

Severity Matrix



Detailed Findings

ID	Title	Severity	Status
M-01	Stuck rewards in first distribution due to delegation activation delay	Medium	Fixed
M-02	Incorrect reward accounting causes unclaimable and stuck rewards	Medium	Fixed
L-01	Revert in reward distribution process in an edge case	Low	Acknowledged
L-02	External operators can register to cap's operator sets without authorization	Low	Fixed
L-03	Incorrect ERC-7201 storage location in EigenServiceManagerStorageUtils	Low	Fixed
I-01	Inefficient validation order in registerStrategy()	Informational	Fixed
I-02	Incorrect NatSpec claims allocation can happen after one block	Informational	Fixed
I-03	Misleading variable name epochDuration causes confusion between time units and epoch counts	Informational	Fixed

Medium Severity Issues

M-01 Stuck rewards in first distribution due to delegation activation delay

Severity: **Medium**

Impact: **Medium**

Likelihood: **Medium**

Files:

[EigenOperator.sol#L63](#)

Status: Fixed

Description: During the operator registration process, the following line is reached:

None

```
File: EigenOperator.sol
63:         IRewardsCoordinator($.rewardsCoordinator).setOperatorAVSSplit(address(this),
msg.sender, 0);
```

The code aims to make the reward split for the operator 0% (default is 10%) so all rewards should go to the restaker. The issue is that the new split will only be activated after a delay of a week due to how EigenLayer functions:

None

```
File: RewardsCoordinator.sol
296:         uint32 activatedAt = uint32(block.timestamp) + activationDelay;
```

During the reward distribution process, this is not considered and the `startTimestamp` of the reward submission can be before the activation delay is over. Thus the operator will be allocated rewards which should not happen and is at the expense of the restakers. Furthermore, the rewards are claimed manually and that functionality is unreachable for the operators, causing them to be stuck.



Recommendations: Gate the first distribution on split activation and clamp the first rewards window to start no earlier than the activation epoch. This ensures no rewards are submitted for a period when restakers could not claim the full rewards amount, preventing unclaimable/stuck funds and unfair reward distribution.

Customer's response: Fixed in [57acOfb](#)

Fix review: Fixed

M-02 Incorrect reward accounting causes unclaimable and stuck rewards

Severity: **Medium**

Impact: **Medium**

Likelihood: **Medium**

Files:

[EigenServiceManager.sol#L411-L418](#)

Status: Fixed

Description: EigenLayer reward submissions have a start timestamp and duration which determines how much a restaker is entitled to, i.e. if they have not staked for the full period, they will be rewarded on a pro rata basis.

The issue is that in the code, the `startTimestamp` of a period can start very shortly after registration which means a restaker is very likely to not be able to delegate before the start timestamp. For example:

1. The current timestamp is a multiple of 86399 (this is the worst case scenario). Let's assume for simplicity that the current timestamp is 86399.
2. There is a registration and creation epoch is 0 as each interval is 86400 seconds.
3. Restaker is very fast to delegate, he manages to do it just an hour after registration at timestamp 89999.
4. In the first reward distribution, the `startTimestamp` is equal to `(creationEpoch + 1) * 86400 = 86400` which is just a second after the registration, it is pretty much impossible for a restaker to delegate during that time if done in a separate transaction. Due to that, the rewards sent to the rewards coordinator will not be all claimable by the restaker.

The unclaimable part will be allocated to the AVS to claim, however no such functionality is available in our contract, thus these funds will remain stuck.



Recommendations: Implement an AVS sweep/claim that credits recovered amounts back into pendingRewards for a future valid window.

Customer's response: Fixed in [c52d172](#)

Fix review: Fixed

Low Severity Issues

L-01 Revert in reward distribution process in an edge case

Severity: Low	Impact: Low	Likelihood: Low
Files: EigenServiceManager.sol# L89-L137	Status: Acknowledged	

Description: During the reward distribution process, we first compute the current epoch:

```
None  
File: EigenServiceManager.sol  
109:     uint32 currentEpoch = uint32(block.timestamp / calcIntervalSeconds);
```

Then, based on it, we compute the start timestamp and the duration which are used for the reward submission:

```
None  
File: EigenServiceManager.sol  
407:     uint256 startTimestamp = _lastDistroEpoch * calcIntervalSeconds;  
408:     uint256 duration = (_currentEpoch - _lastDistroEpoch) * calcIntervalSeconds;
```

If the `currentEpoch` is a result of a perfect division however, then a revert will occur during the `RewardsCoordinator.createOperatorDirectedOperatorSetRewardsSubmission()` flow in `EigenLayer`, specifically in the below check in the internal `_validateOperatorDirectedRewardsSubmission()`:

None

```
File: RewardsCoordinator.sol
488:         require(submission.startTimestamp + submission.duration < block.timestamp,
SubmissionNotRetroactive());
```

This is because the start timestamp added with the duration will result in the current timestamp, causing a revert. Proving it by first seeing what the left side of the check exactly holds:

1. Starting with `startTimestamp + duration`.
2. `lastDistroEpoch * calcIntervalSeconds + (currentEpoch - lastDistroEpoch) * calcIntervalSeconds`
3. `lastDistroEpoch * calcIntervalSeconds + currentEpoch * calcIntervalSeconds - lastDistroEpoch * calcIntervalSeconds`
4. `currentEpoch * calcIntervalSeconds`

Since the current epoch is equal to `block.timestamp / calcIntervalSeconds`, then the 4th step should yield the `block.timestamp` back after the multiplication, thus causing a revert in the `require` statement. However, as we are working in Solidity, then this will only be the case when `currentEpoch` is a result of a perfect division.

Recommendations: As `calcIntervalSeconds` has a value of 86400 in the current `RewardsCoordinator` deployment, the likelihood of this happening is very low as the function must be called exactly at a timestamp divisible by 86400. Due to that, it is recommended to acknowledge the issue as even if the issue were to happen, the reward distribution can be retried the very next second.

Customer's response: Acknowledged

L-02 External operators can register to cap's operator sets without authorization

Severity: Low	Impact: Low	Likelihood: Low
Files: EigenServiceManager.sol#L139-L156	Status: Fixed	

Description: The `registerOperator()` callback function lacks validation to ensure only cap-deployed operators can join operator sets. Any operator registered with EigenLayer can call `AllocationManager.registerForOperatorSets()` and successfully join existing cap operator sets.

The protocol architecture assumes each operator set contains exactly one operator. The protocol creates unique operator sets for each borrower-restaker pair to ensure complete isolation of risk and rewards. However, the `registerOperator()` callback only validates that the AVS matches and the redistribution recipient is correct. It does not verify that the registering operator is a cap-deployed `EigenOperator` contract.

While this does not enable direct financial attacks due to cap's design, it violates core assumptions. The protocol queries collateral and distributes rewards based on specific operators, not operator sets. A malicious operator in the set becomes a mere "phantom member" – registered but ignored by all operations. However, this could cause issues if later functionality is added that iterates over all operators in a set or relies on operator sets instead of operators.

The attack is trivial to execute. Any address can register as an operator with EigenLayer's `DelegationManager` then call `registerForOperatorSets()` with a cap operator set ID obtained from reading contract state..

Recommendations: Add validation in `EigenServiceManager.registerOperator()` to ensure only cap-deployed operators can register by tracking deployed `EigenOperator` addresses and validating against this list in the `registerOperator()` callback function.



Customer's response: Fixed in [ff32d79](#)

Fix review: Fix confirmed

L-03 Incorrect ERC-7201 storage location in EigenServiceManagerStorageUtils

Severity: Low	Impact: Low	Likelihood: Low
Files: EigenServiceManager StorageUtils.sol#L12	Status: Fixed	

Description: EigenServiceManagerStorageUtils uses an incorrect storage location that collides with DelegationStorageUtils. Both contracts use 0x54b6f5557fb44acf280f59f684357ef1d216e247bba38a36a74ec93b2377e200, violating ERC-7201 namespace isolation.

The issue appears to be a copy-paste error where the storage location from DelegationStorageUtils was reused without recalculating it for the different namespace cap.storage.EigenServiceManager.

Proof of Concept:

None

```
$ chisel eval 'keccak256(abi.encode(uint256(keccak256("cap.storage.EigenServiceManager")) - 1)) & ~bytes32(uint256(0xff))'  
Type: uint256  
└ Hex: 0x9813e4033b5f31d05a061ad9d06fb8352756b0443d3cc09baeca467c0811ef00  
└ Hex (full word): 0x9813e4033b5f31d05a061ad9d06fb8352756b0443d3cc09baeca467c0811ef00  
└ Decimal: 68786696764186774574492770130157069297400767117029909338698766047572990750464
```

Recommendations: Update EigenServiceManagerStorageUtils.sol:12 to use the correct ERC-7201 calculated storage location:

None

```
bytes32 private constant EigenServiceManagerStorageLocation =
    0x9813e4033b5f31d05a061ad9d06fb8352756b0443d3cc09baeca467c0811ef00;
```

As a long term solution, consider using the compiler's ability for constant evaluation of expressions at compile time instead of using a constant value.

Customer's response: Fixed in [57ac0fb](#)

Fix review: Fix confirmed

Informational Issues

I-01. Inefficient validation order in registerStrategy()

Description: In `EigenServiceManager.registerStrategy()`, validation checks occur after deploying the `EigenOperator` beacon proxy. This results in unnecessary gas consumption when validations fail as the expensive deployment operation has already been executed.

Recommendation: Move validation checks before the beacon proxy deployment:

None

```
function registerStrategy(...) external {
    // Perform validations first
    if (operatorData.strategy != address(0)) revert AlreadyRegisteredOperator();
    if (operatorData.operatorSetId != 0) revert OperatorSetAlreadyCreated();
    if (IERC20Metadata(...).decimals() < 6) revert InvalidDecimals();

    // Deploy only after validations pass
    address eigenOperator = _deployEigenOperator(_operator, _operatorMetadata);
    operatorData.eigenOperator = eigenOperator;
}
```

Customer's response: Fixed in [O2e3201](#)

Fix review: Fix confirmed

I-02. Incorrect NatSpec claims allocation can happen after one block

Description: The NatSpec for `allocate()` incorrectly states that allocation "needs to wait at least a block" after registration. In reality, operators must wait 17.5 days (ALLOCATION_CONFIGURATION_DELAY) before they can allocate, regardless of their configured allocation delay.

An allocation delay of 0 is correctly set during operator registration. However, EigenLayer's `setAllocationDelay()` does not apply this delay immediately. Instead, it sets `effectBlock`

= block.number + ALLOCATION_CONFIGURATION_DELAY + 1. The delay only becomes active after this effect block is reached which takes approximately 17.5 days on mainnet.

Until the effect block is reached, getAllocationDelay() returns isSet = false, causing modifyAllocations() to revert with UninitializedAllocationDelay(). This means operators cannot allocate for 17.5 days after registration, not "at least a block" as the comment suggests.

Recommendations: Update the NatSpec to accurately reflect the timing requirement:

```
None

/**
 * @notice Allocates the operator set. Can only be called after
ALLOCATION_CONFIGURATION_DELAY
 * (approximately 17.5 days) has passed since registration.
 * @param _operatorSetId The operator set id to allocate
 */
function allocate(uint32 _operatorSetId) external {
    // ...
}
```

Customer's response: Fixed in [02846d9](#)

Fix review: Fix confirmed

I-03. Misleading variable name epochDuration causes confusion between time units and epoch counts

Description: The variable epochDuration has a misleading name that suggests it represents the duration of a single epoch when it actually represents the time interval required between reward distributions.

This naming confusion may contribute to a units mismatch in the code. The variable stores a value in seconds but is used in epoch arithmetic where epochs are counted as integers. The name epochDuration makes it unclear whether the value should be interpreted as "the duration of one epoch" or "the time duration between distributions".



While this is primarily a code clarity issue, it increases the likelihood of implementation and deployment errors.

Recommendations: Rename the variable to clearly indicate its purpose and units, e.g. `epochsBetweenDistributions`.

Customer's response: Fixed in [6e29120](#)

Fix review: Fix confirmed



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.

The review was conducted over a limited timeframe and may not have uncovered all relevant issues or vulnerabilities.

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.