Quantstamp

Covalent - BSPChain & Staking

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Staking, DA blockchain
Timeline	2024-01-23 through 2024-01-29
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README Covalent Network Website
Source Code	• covalenthq/cqt-staking #dc473be
Auditors	• Andy Lin Senior Auditing Engineer • Danny Aksenov Senior Auditing Engineer • Hytham Farah Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	21 Fixed: 8 Acknowledged: 12 Mitigated: 1
High severity findings	2 Fixed: 2
Medium severity findings	6 Fixed: 2 Acknowledged: 4
Low severity findings	7 Fixed: 1 Acknowledged: 5 Mitigated: 1
Undetermined severity findings	0
Informational findings	6 Fixed: 3 Acknowledged: 3

Summary of Findings

Our audit of Covalent's system focused on the `BlockSpecimenProofChain` and `OperationalStaking` contracts. The `BlockSpecimenProofChain` contract allows BSP operators to submit block specimen proofs, while `OperationalStaking` manages validators' staking rewards. The codebase is logically structured, and the README offers clear explanations. Our review also revealed decent test case coverage, although some initial failures need to be addressed.

For this review, we would like to highlight certain limitations regarding the use of the `BlockSpecimenProofChain` results. The system heavily depends on off-chain components, and the documentation for these off-chain component designs is unclear or insufficient, creating ambiguity in our review of some features and the potential impact of issues. Our main concerns lie in the areas of the quorum and audit mechanisms. Additionally, the team has indicated that some aspects of the audit feature are not yet fully implemented, which casts doubt on the system's full production readiness. However, this risk might be acceptable considering the selected set of initial validators.

Fix Review Update: The team addressed the issues we identified as well as those noted by another audit firm. They shared a spreadsheet detailing all their commits, including brief descriptions of the changes, which facilitated our review process. This report concentrates on updates concerning the issues from our initial audit, but we also reviewed fixes for problems reported by another firm.

In our initial review, we identified new issues resulting from these fixes, particularly with COV-1 and COV-7. After communicating these findings to the team, they implemented changes to resolve these issues. This report reflects the final status, with all identified issues, including the newly discovered ones, being addressed. It is important to note, however, that the team has acknowledged the risks associated with blockchain re-organizations, but these risks remain as they have not been fully mitigated.

Second Fix Review: The team added more validations or provided justifications for the issues COV-8, COV-9, and COV-18. We have reviewed the fixes and updated the status of these issues from mitigated to fixed.

ID	DESCRIPTION	SEVERITY	STATUS
COV-1	User Redeem Extra Rewards	High	Fixed
COV-2	Delegators Can Always Obtain Rewards	High	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
COV-3	Unanimous Votes on Reorg Blocks May Still Lead to Pessimistic Phase	Medium	Acknowledged
COV-4	Evenly Split Vote Will Result in Arbitrary Block Choice	Medium	Acknowledged
COV-5	Risk of Single Malicious Operator Triggering Pessimistic Phase	Medium	Acknowledged
COV-6	Conflicting Spec and Implementation Regarding Audit Mechanism	Medium	Acknowledged
COV-7	Delegation Accounting Can Be Compromised	Medium	Fixed
COV-8	Invalid `validatorBitMap` Can Be Consumed by Off-Chain Components	Medium	Fixed
COV-9	Missing Input Validations	Low	Fixed
COV-10	Misleading Event Emission	Low	Acknowledged
COV-11	Misleading View Function for Open Sessions	Low	Acknowledged
COV-12	Risk of Insufficient Gas	Low	Mitigated
COV-13	Variables Never Initialized	Low	Acknowledged
COV-14	Risk of Missing Commission Payouts for Validators	Low	Acknowledged
COV-15	Ownership Can Be Renounced	Low	Acknowledged
COV-16	Silenced `renounceOwnership()`	Informational	Acknowledged
COV-17	Potential Overflow in Session Deadline Calculation	Informational	Acknowledged
COV-18	Potential Issues with Gas Release in `_finalizeWithParticipants()` Function	Informational	Fixed
COV-19	Unfinished Todos	Informational	Fixed
COV-20	Lack of Return Value Checks in Enumerableset Operations	Informational	Acknowledged
COV-21	Misleading Variable Names	Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls

- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

This audit only review two contracts: `OperationalStaking` and `BlockSpecimenProofChain`.

Files Included

- contracts/OperationalStaking.sol
- contracts/BlockSpecimenProofChain.sol

Files Excluded

All other files. Notably, among the excluded files, there is a `BlockResultProofChain` contract, which is quite similar to the `BlockSpecimenProofChain`.

Findings

COV-1 User Redeem Extra Rewards

High Fixed

Update

In commit `90d8adb1`, the team resolved the issue by adjusting the `effectiveAmount` to exactly match the reduced share before recording unstaking data or transferring the tokens for redemption. In other words, the user might receive fewer tokens when calling `unstake()` or `redeemRewards()` due to precision errors, in order to prevent over-granting of funds and maintain the protocol's security.

File(s) affected: `OperationalStaking.sol`

Description: The `OperationalStaking` contract currently manages fractional shares and token amounts in a way that can result in precision loss. This issue emerges due to the rounding down of fractional shares during token redemption (`redeemRewards()`, `redeemAllRewards()`) and unstaking operations (`unstake()`, `unstakeAll()`). As a result, users can accumulate residual shares in the validators' pool, even after redeeming all rewards and unstaking their initial investments. Users can exploit this vulnerability to gain extra shares, allowing them to redeem more tokens than their initial stake or accrued rewards.

Exploit Scenario: The first exploit scenario demonstrates how users can leverage the precision error to unstake and redeem tokens based only on their principal investment:

1. Alice delegates her stakes for 1000 tokens when the exchange rate is 1.1. She will get 909 shares.
2. Alice unstake just 2 tokens. This will burn `floor(2 / 1.1) = floor(1.8) = 1` shares. Alice unstakes 2 tokens for 500 times, each unstake will reduce 1 of her share. Alice will still have 909 - 500 = 409 shares and zero stakes.
3. Alice call `redeemAllRewards()` to redeem the money from her remaining 409 shares. This will be possible as her staking amount is zero now. She will redeem `409 * 1.1 = 449` tokens.

4. Alice call `recoverUnstaking()` for the 500 unstakes. She will have staking amount as 1000 with 500 shares. This re-units her segmented unstakes.
5. Alice `unstakeAll()` first and then call `recoverUnstaking()` for just 550 token amount. This will give her `550 / 1.1 (exchange rate) = 500 shares` with staking amount 550 and 450 in the unstaking queue.
6. Again, Alice repeats the previous step to unstake 2 tokens each, which will reduce 550 / 2 = 275 shares, and she can redeem `500 - 275 = 225 shares`. And this will give her `225 * 1.1 = 247 tokens`.
7. Alice re-units her segmented 275 unstakes back to have 550 tokens in stake with 275 shares.
8. Alice call `unstakeAll()` again. With the 550 unstaking tokens, she calls `recoverUnstaking()` for 303 tokens. Now she will have 303 token in stake, 275 shares, and one two unstakings in queue: 450 (previous one) and `550 - 303 = 247` for this round.
9. Until now, Alice has already redeemed 449 + 247 = 639 tokens while the remainings are either in the unstake queue or still in stake. Each time the shares will cut to half, so she can at most redeem the same principle amount of tokens out if she continues the steps

The second scenario illustrates how users can accrue additional shares through a series of strategic redemptions and unstakings:

1. Initially, a validator stakes 9000 tokens, setting the exchange rate at 1 and owning 9000 shares.
2. After several reward rounds, the exchange rate increases to 10.
3. Alice stakes 10000 tokens at an exchange rate of 10, receiving 1000 shares.
4. Post Alice's staking, a 1000-token reward distribution raises the exchange rate from 10 to 10.1. Alice's shares now equate to 10100 tokens, and the validator's to 90900 tokens.
5. Alice redeems 20 tokens, decreasing her shares to 999 due to rounding (`20 / 10.1 = 1.98 -> sharesToBurn = 1`).
6. Alice redeems another 20 tokens four times, reducing her shares to 995. By now, all 100 reward tokens are redeemed.
7. Alice unstakes her original 10000 tokens, removing 990 shares (`10000 / 10.1 = 990.09`), leaving 5 residual shares.
8. Despite redeeming all rewards (100) and unstaking her initial investment (10000), Alice retains 5 shares, enabling further reward accrual and diluting other stakers' rewards.
9. More critically, Alice can invoke `redeemAllRewards()` to withdraw an unauthorized 50 tokens (`5 shares * 10.01`) from the validator's reward pool. If the validator attempts to redeem and unstake all funds, they risk failure due to an insufficient token balance.

Recommendation: When users withdraw funds, it is preferable to over-burn shares rather than maintain precision loss to ensure the pool's adequate fund availability. A proposed solution is modifying the `_tokensToShares()` function to include a parameter for rounding direction:

```
function _tokensToShares(uint128 amount, uint128 rate, bool roundUp) internal pure returns (uint128) {
    uint256 precisionControl = 0;
    if (roundUp && (uint256(amount) * DIVIDER) % uint256(rate) != 0) {
        precisionControl = 1;
    }
    return uint128((uint256(amount) * DIVIDER) / uint256(rate) + precisionControl);
}
```

Then, set the `roundUp` flag to true in `_unstake()` and `_redeemRewards()` functions to prevent unintended extra rewards.

COV-2 Delegators Can Always Obtain Rewards

High Fixed

Update

The team fixed the issue by adding another cooldown period (`recoverUnstakingCoolDown`) has been added to the code to avoid front-running scenarios in the commits: `e7ab9ab7eb` and `5aae79d4`.

File(s) affected: `OperationalStaking.sol`

Description: The `OperationalStaking` contract enables the staking manager to distribute rewards to validators using the `rewardValidators()` function. However, delegators can exploit this by front-running the transaction, quickly performing an `unstake()` followed by a `redelegateUnstaked()` to shift their delegation to validators who receive the best rewards.

Exploit Scenario: Consider the following example:

1. Alice initially stakes with validator Bob.
2. The staking manager initiates a call to `rewardValidators()`, choosing to reward only Carol.
3. Alice, monitoring the mempool, notices this and immediately calls `unstake()` followed by `redelegateUnstaked()` to Carol to claim her share of the rewards.

Recommendation: To mitigate this issue, consider implementing a restriction where `redelegateUnstaked()` can only occur a certain number of blocks (even one is good) after the `unstake()` function. This delay would prevent immediate re-delegation and reduce the likelihood of such exploitation.

COV-3 Unanimous Votes on Reorg Blocks May Still Lead to Pessimistic Phase

Medium Acknowledged

> **Update**
> The team acknowledged that they will proceed, accepting the risk associated with some unaudited specimens, given the low reorganization rate.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: The number of contributors is tallied across all the `blockHashes`, and quorum is based on whether there is a specimen hash with a majority of votes. However, in the case of a reorg, votes will be naturally split among the different blocks, even though there may be unanimity in the votes for block specimen hashes for each block hash.

From our current understanding, the "auditor" mechanism has not been implemented yet. Therefore, it is likely that validators will lose their reward if a session enters the pessimistic phase, because the auditor cannot yet prove their validity.

Exploit Scenario: Suppose there are three reorg blocks A, B, and C, and all validators spot it and report the same specimen hashes A', B', and C', respectively. In this case, each hash receives 33% of the votes, which is less than the quorum of 50%, even though they are all reported unanimously.

Recommendation: The team should clarify if the risk is acceptable and consider some of the following mitigations:

1. Currently, `contributorsN` is calculated as the sum of participants across all hashes. It might be more suitable to divide this number by either the total number of unique BSP operators or the total number of unique participants for a given block hash. Alternatively, a two-layer mechanism could be implemented: firstly, assess the legitimacy of a block hash based on obtaining enough votes from all BSP operators, and then, evaluate the winning specimen hash, ensuring it has received more than 50% of the votes from participants who voted for specimens within that block hash.
2. In the mid to longer term, implement the auditor mechanism to ensure a smooth resolution for the pessimistic phase.

COV-4
Evenly Split Vote Will Result in Arbitrary Block Choice

Medium | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > reorgs are minimal in ethereum PoS; even so we expect the voting mechanism to be able to handle that. We accept that some blocks might remain un-audited in the rare case.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: In the scenario where there is an exact 50/50 split in the vote between two hashes, the `BlockSpecimenProofChain.finalizeSpecimenSession()` function will arbitrarily pick one of the two blocks as the finalized blocks. To be more specific, with the current implementation, the `specimenHash` that submits later will win the tie now, but this approach may not always reflect the most accurate or representative choice, leading to potential fairness and security concerns in the decision-making process.

Exploit Scenario: Suppose there are 4 validators and two vote on blockhash `A` with specimen hash `A'` and two vote on blockhash `B` with specimen hash `B'`. Assuming this meets the threshold of the `_minSubmissionsRequired`, the `finalizeSpecimenSession()` will choose whichever blockhash occurs first in `BlockHashesRaw`. Even though, in principle, this choice is completely arbitrary.

Recommendation: The team should clarify whether it is acceptable for any of the multiple valid specimen hashes to be the winning hash when there are several valid choices. Additionally, the team should consider designing or applying a tie-breaker mechanism if needed.

COV-5
Risk of Single Malicious Operator Triggering Pessimistic Phase

Medium | Acknowledged

> **Update**
> The team acknowledged the issue with the following statement:
>
> > schema upgrades are rare; and even so, the releases from our side are done way before the new fields etc. reach ethereum mainnet.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: We observed that even just a single malicious operator can cause a session to enter the pessimistic phase during a schema update. Discussions with the team revealed that when the specimen block schema updates, different BSP operators might be running block generation using different versions of the block schema, leading to good operators generating divergent specimen block hashes. Normally, the specimen

block hash should be deterministic from a source block's data. However, when operators diverge into different groups, a single malicious operator could force the session into the pessimistic phase.

Here is the proof of why a single malicious operator can trigger a pessimistic phase. Given a total number of `n` operators, with 1 being malicious, there are `n-1` good operators. During a schema upgrade, this can split the good operators into two groups, with each group getting `(n-1)/2` votes for its specimen block hash. Assuming that the malicious operator votes for another self-created invalid hash, this will result in three hashes:

1. Valid specimen hash 1 for the old schema: `(n-1)/2` votes.
2. Valid specimen hash 2 for the new schema: `(n-1)/2` votes.
3. Invalid hash: 1 vote.

Even with just 1 vote, both valid schemas will fail to pass the quorum of `n / 2` and thus enter the pessimistic phase.

Furthermore, the Covalent team mentioned that the auditor mechanism is not fully implemented yet, and thus not utilizing the auditor yet. In other words, a single malicious operator can force a session to end up without any rewards and also affect other potential outcomes when there is no canonical block hash for a session.

To examine this further, since each operator can submit up to `maxSubmissionsPerBlockHeight` (referred to as `max_submission` here) block hashes, this allows a malicious operator to further dilute the votes. Here is a proof that takes more variables into consideration. Given a total number of `n` operators, and `f` of those are malicious, there are `n-f` good operators. The good operators are split into two groups during schema changes, each with `(n-f)/2` members. The scenarios can become:

1. Valid hash for the old schema: `(n-f)/2` votes.
2. Valid hash for the new schema: `(n-f)/2` votes.
3. Invalid hash 1: `f` votes.
4. Invalid hash 2: `f` votes.
5. ...Invalid hash `max_submission` : `f` votes.

The `contributorsN` in this case will be `(n-f)/2 + (n-f)/2 + max_submission * f = n + (max_submission - 1)*f`. Given the Quorum `Q`, for one of the valid hashes to pass the quorum, it requires that:

```
((n-f) / 2) / contributorsN > Q

--> ((n-f) / 2) / (n + (max_submission - 1)*f) > Q
--> (n-f) / 2 > Q*n + Q*(max_submission - 1)*f
--> n - f > 2*Q*n + 2*Q*(max_submission - 1)*f
--> n - 2*Q*n > 2*Q*(max_submission-1)*f + f
--> n*(1-2*Q) > (2*Q*(max_submission-1)+1)*f
--> n > f * ((2*Q*(max_submission-1)+1) / (1 - 2*Q))
```

For instance, with a `max_submission` of 3 (as deployed by the team), and a quorum lowered to 20% (Q = 0.2), it will need `n > f * ((2 * 0.125 * (3 - 1) + 1) / (1 - 2 * 0.125))`, thus `n > 3*f`. This implies that the fault threshold is around 33% when the quorum is lowered to 20%, while it is close to zero when Q approaches 50%.

From the above observations:

1. When the quorum is `>= 50%`, the fault tolerance can be zero or it may be impossible to meet the quorum during an upgrade (network partition).
2. Counterintuitively, the lower the quorum, the higher the fault tolerance for triggering the pessimistic phase. However, this assumes that "good BSP operators" will always submit their specimen hashes. It is unclear to us if there is an incentive to ensure this action as well. The balance should be carefully designed.

Ultimately, choosing a quorum level during a network partition involves a tradeoff. Setting a higher quorum (>=50%) enhances consistency among operators but increases the risk of entering the pessimistic phase more easily, potentially disrupting certain features. Conversely, a lower quorum increases the likelihood of obtaining a winning specimen hash that meets the quorum threshold. However, in cases where multiple hashes meet the quorum requirement, it's unclear which one should be used or how to break ties (this issue is addressed in another section of this report). This approach also risks lacking a dominant consensus on a specific hash.

Exploit Scenario: Here is a sample scenario:

1. Given 13 BSP operators in total: 6 running with old schema, 6 running with new schema, and 1 malicious operator.
2. When they vote on a session that is just on the schema update, there will be three hashes in vote, and get 6, 6, 1 votes respectively. Both of the hashes that got 6 votes failed to pass the quorum check as it require more than 13 / 2 = 6.5 votes.
3. `QuorumNotReached` event is emitted.

Recommendation: Here are some of the directions that can help mitigate the issue:

1. The team should consider a short-term solution for sessions reaching the pessimistic phase. Or, the team should accept the associated risks, such as the potential loss of rewards for good validators.
2. Currently, `contributorsN` is set to the sum of participants in all hashes. Consider dividing by either the total number of unique BSP operators or the total number of unique participants in a given session instead. This will reduce the dilute factor of the votes impacted by `maxSubmissionsPerBlockHeight`.
3. The team should decide what is more important for their business use cases. Lowering the quorum may ensure some fault tolerance in network partition scenarios if the team prefers to always have a 'winning hash' that passes the quorum. However, if the team values consistency across the network and finds the risk of entering the pessimistic phase acceptable, maintaining the current quorum threshold (50%) is advisable.
4. Mid to long term, the team should fully implement the auditor mechanism to mitigate this risk.

COV-6
Conflicting Spec and Implementation Regarding Audit Mechanism

Medium | Acknowledged

> **Update**
> The team acknowledged the issue and stated that the lack of conflict-resolving auditor functionality is considered part of the contract for now. Note that the code has been updated to rename `requiresAudit` to `isSessionDone` to avoid confusion.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: The `BlockSpecimenProofChain.finalizeSpecimenSession()` function always sets the `requiresAudit` field of the `sessions` struct to `true` regardless of the results of the session, even if a quorum reaches.

This contradicts the README description in the "Proof Chain Explained" section:

> An Auditor is an operator who arbitrates sessions that did not reach quorum.

and potentially the case 1 example of the README doc:

> **Case 1 - Quorum achieved, only Optimistic Phase**
>
> ....skip
>
> **Pessimistic phase**
> Not required since the quorum was achieved and no other block hashes submitted.

However, the README also has the following description in the "Proof Chain Explained":

> If quorum is achieved the participants who submitted the agreed specimen hashes get rewarded. The rewards are pushed from the ProofChain to the Staking. If quorum was not achieved nothing happens. In both cases at the end of the transaction, the session gets marked as `requires audit`.

This aligns with the current implementation but contradict with other part of the doc. Meanwhile, from our discussion with the team, the audit mechanism is not in use yet for the off-chain part.

Recommendation: The team should clarify and unify the spec. Depending on the desired spec, the team should align the contract implementation.

If an audit is only required when a quorum is not reached, `session.requiresAudit` should only be set in cases where no quorum is achieved. However, some of the conditional checks that depend on `requiresAudit` will also need to be revisited to ensure that the assumptions still hold.

COV-7 Delegation Accounting Can Be Compromised

Medium | Fixed

> **Update**
> The team fixed the issue by introducing a new `checkDelegatorExists()` function and applying the check in the `setValidatorAddress()` function. See commits: `d89d3e21` and `d38dd16`.

File(s) affected: `OperationalStaking.sol`

Description: The `OperationalStaking.setValidatorAddress()` function permits validators to update their effective address to a new one. However, if the `newAddress` is already an existing delegator of the validator, this could disrupt the `v.delegated` accounting logic. Transferring the validator address to a delegated address results in the previously delegated amount remaining in the `v.delegated` accounting. However, when unstaking occurs, the `_unstake()` function will not reduce the `v.delegated` value, as it will now be considered unstaking from the validator, not a delegator. This situation can also prevent the validator from unstaking as desired due to the `v.delegated <= newValidatorMaxCap` constraint.

Recommendation: It is advisable to include a check in the `setValidatorAddress()` function to prevent transferring to existing delegators. For example, the function could verify that `v.stakings[newAddress].staked == 0` at the beginning to ensure that the new address is not already a delegator.

COV-8
Invalid `validatorBitMap` Can Be Consumed by Off-Chain Components

Medium | Fixed

> **Update**
> The team added the missing validation to the `BlockSpecimenProofChain.enableValidator()` in commit `f2723ac`. Now, all the recommended validations have been added.

> **Update**
> The team added the check to the `OperationalStaking.addValidator()` function in commit `d382f66b`. However, the check has not yet been added to the `BlockSpecimenProofChain.enableValidator()` function.

File(s) affected: `BlockSpecimenProofChain.sol`, `OperationalStaking.sol`

Description: The `BlockSpecimenProofChain._finalizeWithParticipants()` function creates a bitmap for the validators using the expression `validatorBitMap |= (1 << (255 - validatorIDs[participant]))`. This calculation assumes that the validator ID is at most 255. However, this assumption is not enforced in the contract.

1. `BlockSpecimenProofChain.enableValidator()`: This function does not check whether the input `validatorId` is `<= 255`.
2. `OperationalStaking.addValidator()`: This function does not verify if `validatorsN` stays within the 255 limit. Moreover, it relies on an incremental counter that cannot be decreased. Once a validator is added, the validator ID space is occupied and cannot be released. The validator can only be enabled or disabled later to change its effectiveness.

We believe the staking manager relies on the `validatorBitMap` to determine the validators to reward in the `OperationalStaking.rewardValidators()` function. Therefore, we consider the impact of an incorrect `validatorBitMap` relatively high, and we have escalated it to medium severity.

Exploit Scenario: The staking manager accidentally adds too many validators, causing the `BlockSpecimenProofChain._finalizeWithParticipants()` function to emit an incorrect `validatorBitMap` in the `BlockSpecimenQuorum` event, potentially leading to the rewarding of incorrect validators.

Recommendation: As a short-term solution, consider adding a few validations to enforce the assumption regarding the size of the validator set:

1. In the `BlockSpecimenProofChain.enableValidator()` function, add a validation that `validatorId <= 255`.
2. In the `OperationalStaking.addValidator()` function, ensure that `validatorsN <= 255` before incrementing it (`validatorsN += 1`).

However, this fix for the `OperationalStaking` contract implies that there may be a limit on the number of validators in the future if the validator set changes. A shortage of IDs may occur, requiring a more substantial design change to address this issue adequately.

COV-9 Missing Input Validations

Low · Fixed

> **Update**
> After the first fix-review report upate, the team updated us with some additional validations being added and provided explanations for those they decided not to include:
>
> - **BlockSpecimenProofChain**
> - `removeAuditor()`: the existing `require(...)` validation will fail in the case of a zero address.
> - `removeGovernor()`: the existing `require(...)` validation will fail in the case of a zero address.
> - `setBlockSpecimenSessionDuration()`: if `sessionDeadline` is set too high, the system will not have any finalized sessions and will halt. Thus, the team considers it a low priority and prefers to go without validation to save gas.
> - (fixed) `setMaxSubmissionsPerBlockHeight()`: validation added.
> - (fixed) `setBlockSpecimenReward()`: an upper limit has been added.
> - **OperationalStaking**
> - (fixed) `initialize()`: added the validation that `cqt` cannot be zero. For the cooldown values, having them be zero is possible.
> - (fixed) `recoverUnstaking()`: validation added.
> - (fixed) `redelegateUnstaked()`: validation added.
> - `setValidatorCoolDown()`: the cooldown period can be set to zero.
> - `setDelegatorCoolDown()`: the cooldown period can be set to zero.
>
> All validations that we listed but were not added have sufficient justification; thus, we are updating the status to fixed after reviewing the latest validation status on the commit `16c6b2b`.

> **Update**
> From the code in the fix-review commit `0352f75d60` (note: this commit did not add the validations), the team implemented some of the recommended validations, but not all. The following list identifies functions that are still missing validations after the fix:
>
> - **BlockSpecimenProofChain**
> - `removeAuditor()`
> - `removeGovernor()`
> - `setBlockSpecimenSessionDuration()`
> - `setMaxSubmissionsPerBlockHeight()`
> - `setBlockSpecimenReward()`
> - **OperationalStaking**
> - `initialize()`
> - `recoverUnstaking()`
> - `redelegateUnstaked()`
> - `setValidatorCoolDown()`

- `setDelegatorCoolDown()`

File(s) affected: `BlockSpecimenProofChain.sol`, `OperationalStaking.sol`

Description: Input validation is crucial for maintaining system security and reliability. It helps to prevent errors and potential attacks by restricting the range of inputs that attackers can use. We recommend adding input validations in the following areas:

BlockSpecimenProofChain

1. `initialize()`: `initialOwner` is missing a non `0x` address validation check.
2. `addAuditor()`: `auditor` is missing a non `0x` address validation check.
3. `removeAuditor()`: missing validation to ensure that `auditor` is not a zero address.
4. `addGovernor()`: `governor` is missing a non `0x` address validation check.
5. `removeGovernor()`: missing validation to ensure that `governor` is not a zero address.
6. `setNthBlock()`: `n` is missing a non zero check.
7. `setBlockSpecimenSessionDuration()`: `newSessionDuration` + `block.number` should be validated to not exceed max `uint64`. In addition, `newSessionDuration` is missing a non zero value check.
8. `setMaxSubmissionsPerBlockHeight()`:
   - `chainId` should be checked to be an existing chain id.
   - `maxSubmissions` should be validated to be in a valid range.
9. `setBlockHeightSubmissionsThreshold()`: `threshold` should be validated to be non zero value.
10. `setSecondsPerBlock()`: `secondsPerBlock` should be validated to be in a valid range.
11. `submitBlockSpecimenProof()`: `blockHash`, `specimenHash`, and `storageURL` should be validated to be non-empty values.
12. `setQuorumThreshold()`: This function is missing validation to ensure that the `quorum` is within a valid range (e.g., between 0 and 10^18). Additionally, if a minimal threshold is preferred, validation for the minimal threshold should also be included.
13. `setBlockSpecimenReward()`: missing validation to ensure that `newBlockSpecimenReward` is within a valid range.
14. `setBlockSpecimenSessionDuration()`: missing validation to ensure that `newSessionDuration` is within a valid range.
15. `setMinSubmissionsRequired()`: missing validation to ensure that `minSubmissions` is within a valid range.
16. `submitBlockSpecimenProof()`: consider validating that `storageURL` is not empty unless it is intended to be an optional input.

OperationalStaking

1. `initialize()`: missing input validation on `cqt`, `vCoolDown`, `dCoolDown`, `maxCapM`, `vMaxStake`.
2. `setValidatorMaxStake()`: `maxStake` should be greater than or equal to `minStake`.
3. `recoverUnstaking()`: `amount` should be validated to be a non zero value.
4. `transferUnstakedOut()`: `amount` should be validated to be a non zero value.
5. `redelegateUnstaked()`: `oldValidatorId` should not be equal to `newValidatorId`.
6. `setValidatorCoolDown()`: missing validation to ensure that `coolDown` is greater than zero.
7. `setDelegatorCoolDown()`: missing validation to ensure that `coolDown` is greater than zero.

Recommendation: Consider implementing the suggested validations as detailed in the description above.

COV-10 Misleading Event Emission

Low | Acknowledged

Update

The client acknowledged the issue with the following explanation:

> state change is not necessary for that event to be emitted

File(s) affected: `BlockSpecimenProofChain.sol`

Description: Both `disableValidator()` and `enableValidator()` do not perform a sanity check on the status of the `_validatorEnabled` prior to updating the state. This can lead to a potentially confusing scenario, where the `ValidatorDisabled` and `ValidatorEnabled` events are emitted without accurately reflecting the state change.

Recommendation: Consider only performing the state change and event emission, if the validators state differs from the intended change.

COV-11 Misleading View Function for Open Sessions

Low | Acknowledged

Update

The client acknowledged the issue with the following explanation:

> isSessionOpen is meant for bsp-agent and refiner (other parts of Covalent network). As such the implementation shouldn't check for the block ranges. (when submitSpecimen is called, the range check happens and out of bounds is thrown, enough of them and we need to update the chaindata)

Description: The `BlockSpecimenProofChain.isSessionOpen()` function returns true if the submission limit is not exceeded and the current block is before the deadline. It does not take into account sessions that cannot happen because they are outside of the accepted

window, (whether or not it is some future date and eventually will happen, or past date and will never happen).

The "session window" logic can be referred to the following code in the `submitBlockSpecimenProof()` function:

```
uint256 currentBlockOnTargetChain = cd.blockOnTargetChain + (((block.number - cd.blockOnCurrentChain) *
_secondsPerBlock) / cd.secondsPerBlock);
            uint256 lowerBound = currentBlockOnTargetChain >= cd.allowedThreshold ?
currentBlockOnTargetChain - cd.allowedThreshold : 0;
            require(lowerBound <= blockHeight && blockHeight <= currentBlockOnTargetChain +
cd.allowedThreshold, "Block height is out of bounds for live sync");
```

Recommendation: Return `false` if a session is outside of the current session window as well.

COV-12 Risk of Insufficient Gas

Low Mitigated

> **Update**
> The team added validation to ensure the max number of validator cannot exceed 255 in the commit `d382f66` . Also, they expect to run it with just 20 validators in practice.

Description: The `BlockSpecimenProofChain.submitBlockSpecimenProof()` function includes a nested for-loop that ensures the same validator does not submit different specimen proofs for the same block hash. Although the code comment suggests a maximum of 10 iterations, the worst-case scenario could involve each validator submitting unique specimen hashes, leading to gas costs on the order of `O(num_validators^2)` .

Similarly, the `finalizeSpecimenSession()` function executes a nested for-loop with a complexity of `O(blockSpecimenHashesLength * specimenHashes.length)` . Since a single validator can submit a maximum of `cd.maxSubmissionsPerBlockHeight` block hashes per session, the complexity is effectively `O(cd.maxSubmissionsPerBlockHeight * num_validators)` .

Team discussions reveal that the number of validators will be capped at 255, and a typical value for `maxSubmissionsPerBlockHeight` is likely to be around 3. This leads to the classification of the issue as low severity. Also, the validators are more incentives to submit the valid proofs. However, it is important to note that the current contract does not enforce a limit on the size of the validator set.

Recommendation: Consider the following mitigations:

1. Implementing a cap on the maximum size of the `specimenHashes` array within the `submitBlockSpecimenProof()` function to mitigate the risk of excessive gas consumption.
2. In `setMaxSubmissionsPerBlockHeight()` function, add a cap validation to the `maxSubmissionsPerBlockHeight` input.
3. In `enableValidator()` function, add a validation that `validatorId <= 255` .

COV-13 Variables Never Initialized

Low Acknowledged

> **Update**
> The team clarified that `bspRequiredStake` is a deprecated variable but keeping it to keep the storage layout for the upgraded contract. We suggest add a comment on the variable to make it easier to understand the reason of keeping it.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: The `BlockSpecimenProofChain` contract includes the `_bspRequiredStake` storage variable, which is never initialized and lacks a setter method. Consequently, the first return value of the `getBSPRoleData()` function is perpetually zero.

Recommendation: Consider either removing this variable or initializing it in the `initialize()` function. Alternatively, creating a setter function for it could be a viable solution.

COV-14 Risk of Missing Commission Payouts for Validators

Low Acknowledged

> **Update**
> The team acknowledged the issue and decided to not fix it due to its low severity.

File(s) affected: `OperationalStaking.sol`

Description: The current implementation in `OperationalStaking.sol` risks validators not receiving commission payouts due to the way Solidity handles rounding. This issue arises in the following snippet:

```
commissionPaid = uint128((uint256(amount) * uint256(v.commissionRate)) / DIVIDER);
```

In scenarios where the product of `amount` and `v.commissionRate` is less than `DIVIDER`, the division result gets rounded down to zero. This could lead to instances where validators receive no commission, particularly if the `stakingManager` calculates low reward amounts or if the commission rate itself is minimal. The exact mechanics of reward distribution by `stakingManager` remain unclear, but this potential rounding issue could result in validators being under-compensated.

Recommendation: Implement a safeguard to ensure that `commissionPaid` is always greater than zero, thus preventing the omission of deserved commissions for validators.

COV-15 Ownership Can Be Renounced

Low Acknowledged

> **Update**
> The client clarified that this is intended and also protected with multi-signatures.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be executable. In the `BlockSpecimenProofChain` contract, `addGovernor()` and `removeGovernor()` functions will be at risk due to the renouncement of ownership.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

COV-16 Silenced `renounceOwnership()`

Informational Acknowledged

> **Update**
> The client clarified that they actually want to keep the `renounceOwnership()` function working and they have removed the function overriding.

File(s) affected: `OperationalStaking.sol`

Description: The team has overridden the `OperationalStaking.renounceOwnership()` function to be an empty function. Although we typically suggest overriding the function to prevent situations where the owner of the contract is removed, we recommend having the function revert as well to avoid successful transactions leading to misleading interpretations.

Recommendation: Consider reverting in the function instead of leaving it empty.

COV-17 Potential Overflow in Session Deadline Calculation

Informational Acknowledged

> **Update**
> The team considered this as low priority. The chance of occurrence is indeed low as long as `_blockSpecimenSessionDuration` assignment is properly handled.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: The following line of code in `submitBlockSpecimenProof()`:

```
session.sessionDeadline = uint64(block.number + _blockSpecimenSessionDuration);
```

is potentially vulnerable to an overflow error. This occurs because `block.number`, which is a `uint256`, is being added to `_blockSpecimenSessionDuration` and then cast to a `uint64`. An overflow could occur if the sum exceeds the maximum value that a `uint64` can hold.

We understand that in practice, it is very unlikely to break. Thus, we only set the severity as informational. However, we want to highlight that `_blockSpecimenSessionDuration` is not capped. So, the risk is not necessarily zero.

Recommendation: Consider setting a practical cap to the `_blockSpecimenSessionDuration` value.

COV-18
Potential Issues with Gas Release in `_finalizeWithParticipants()` Function

Informational | Fixed

Update
On the second fix review, the team fixed the second issue by removing the `pd.stake = 0` line in commit `ac095d0`.

Update
The team fixed the first issue in the description by changing the condition to `pd.submissionCounter > 0` in the commit `c8184f4903`.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: At the end of `_finalizeWithParticipants()`, the participant's data is being reset as a way to release gas with the following code:

```
if (pd.submissionCounter == 1) {
    pd.submissionCounter = 0;
    pd.stake = 0;
}
```

However there is some confusion around this implementation:

1. Currently, gas is being released for participants who have only had one submission during the session. This seems to be an error, as participants can end up with more than one submission per session.
2. According to the code comments for the `SessionParticipantData` struct, the stake value has been deprecated and will now always remain 0. It seems that assigning 0 to `pd.stake`, which may already be 0 will actually result in a consumption of gas and not a refund as may be anticipated.

Recommendation: We recommend the following changes:

1. `pd.submissionCounter >= 1`
2. either refrain from assigning 0 to `pd.stake` or check that the value is indeed non-zero before zero'ing it out.

COV-19 Unfinished Todos

Informational | Fixed

Update
The team cleared the session storage in the commit `4a7baa8e54`.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: The `BlockSpecimenProofChain.finalizeSpecimenSession()` function contains the comment:

```
// TODO: doesn't free session space. Though it should.
```

This might indicate that additional steps are required for production readiness or before a code freeze, particularly in optimizing storage management and enhancing contract efficiency.

Recommendation: Consider completing the TODO or removing the comment if it is no longer part of the planned tasks.

COV-20
Lack of Return Value Checks in Enumerableset Operations

Informational | Acknowledged

Update
The team considered this as a low priority fix and has decided to keep it as is.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: The `BlockSpecimenProofChain` contract performs several operations on `EnumerableSet` variables. OpenZeppelin's `EnumerableSet` library is designed to not fail on `add()` or `remove()` operations even if the items already exist or do not exist before

execution. Instead, it returns a boolean to indicate whether an actual action has been taken. Not checking these return values can be a problematic pattern, particularly for the `remove()` function, as it might result in no removal and inadvertently trigger subsequent actions.

While no exploits were identified during the audit, as all functions that operate on the `EnumerableSet` have pre-validation against `operatorRoles`, it is still recommended to implement checks to avoid potential bugs in future code modifications.

Recommendation: Ensure to check the return values of the `remove()` and `add()` functions on variables like `_roleNames`, `_blockSpecimenProducers`, `_governors`, and `_auditors`.

COV-21 Misleading Variable Names

Informational Fixed

Update
The team changed the variable name as suggested in the commit `0352f75d60`.

File(s) affected: `BlockSpecimenProofChain.sol`

Description: The `BlockSpecimenProofChain.initialize()` function has the input variable `initialOwner` which might be misleading. The role of the `initialOwner` is the first member of the governor and not the contract owner. Given the current setup of the `OwnableUpgradeable`, the `BlockSpecimenProofChain` contract's owner will be initiated to the deployer (`msg.sender` of `initialize()`).

Recommendation: Consider renaming the variable name to `initialGovernor` instead.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither v1.0.1

Steps taken to run the tools:
Run Slither from the project directory: `slither .`.

Note that this is a Quantstamp internal forked version of the original Slither. We currently forked from the `v0.9.3` version.

Automated Analysis

Slither

Slither identified 178 results, most of which were false positives or out of the scope of this audit. We have incorporated the valid findings into the report's issues.

Test Suite Results

The current test is run using `npx hardhat coverage`.

Update: After the fix review, we re-run the unit tests in the commit `1ea6c5`:

- `npx hardhat coverage --testfiles test/cqt-staking/unit-tests`
- `npx hardhat coverage --testfiles test/block-specimen-proof-chain/unit-tests`

```
Version
=======
> solidity-coverage: v0.8.10

Instrumenting for coverage...
============================

> BlockResultProofChain.sol
> BlockSpecimenProofChain.sol
> CovalentQueryTokenFaucet.sol
> ERC20Permit/ERC20Permit.sol
> ERC20Permit/IERC2612Permit.sol
> IOperationalStaking.sol
> OperationalStaking.sol

Compilation:
============

Nothing to compile

Network Info
============
> HardhatEVM: v2.19.5
> network:    hardhat



  Tests addAuditor()
    ✔ Lets governor add an auditor and emits OperatorAdded event with correct args
    ✔ Reverts when non-governance address adds an auditor (59ms)
    ✔ Should set correct role (49ms)
    ✔ Should be able to add multiple auditors (85ms)
    ✔ Should revert when trying to add auditor who is an operator (67ms)

  Tests addGovernor()
    ✔ Lets owner add a governor and emits OperatorAdded event with correct args
    ✔ Reverts when non-owner address adds a governor
    ✔ Should set correct role (46ms)
    ✔ Should be able to add multiple governors (79ms)
    ✔ Should revert when trying to add a governor who is an operator (65ms)

  Tests addBSPOperator()
    ✔ Lets governance address add bsp operator
    ✔ Emits OperatorAdded event (43ms)
    ✔ Reverts when non-governance address preapproves an address for a role type
    ✔ Reverts when adds the same operator twice (90ms)
    ✔ Sets operators under correct validator (137ms)
    ✔ Operator is disabled after being added (66ms)
    ✔ Should set correct validator id (149ms)
    ✔ Should set correct operator role (144ms)
    ✔ Should set correct validator id (141ms)

  Tests disableValidator()
    ✔ Lets a governance role disable a validator after they are added (131ms)
    ✔ Emits ValidatorDisabled when a validator is disabled (46ms)
    ✔ Reverts when non-stakingmanager tries to disable a validator (46ms)

  Block Specimen Session finalization Tests
    ✔ Reverts if the block specimen session has not started (55ms)
    ✔ Reverts if the deadline has not been reached (79ms)
    ✔ Changes require audit to true when not enough participants submitted, emits event and reverts if
called again (278ms)
```

```
    ✔ Changes require audit to true when quorum was not reached, emits event and reverts if called again
(727ms)
    ✔ Emits specimen hash reward awarded event with the correct args when quorum is achieved (813ms)
    ✔ Emits specimen hash reward awarded event with the correct args when quorum is achieved (768ms)
    ✔ Emits specimen hash reward awarded event with the correct args when quorum is achieved (752ms)

  Tests submitBlockSpecimenProof()
    ✔ Lets a BSP role submit a specimen proof and add matching urls (239ms)

  Tests remove auditor
    ✔ Lets Governance remove an audior (88ms)
    ✔ Emits OperatorRemoved (62ms)
    ✔ Does not let a non-governance role call remove auditor (90ms)
    ✔ Should revert when trying to remove an auditor that does not exist or has a different role (87ms)

  Tests remove governor
    ✔ Lets Governance remove an audior (89ms)
    ✔ Emits OperatorRemoved (59ms)
    ✔ Does not let a non-owner role call remove governor (80ms)
    ✔ Should revert when trying to remove an governor that does not exist or has a different role (82ms)

  Tests Governance control: removeBSPOperator()
    ✔ Lets Governance remove an operator (39ms)
    ✔ Emits OperatorRemoved (44ms)
    ✔ Does not let a non-governance role call removeBSPOperator()
    ✔ Does not emit ValidatorDisabled on proofchain contract on inappropriate role (54ms)
    ✔ Removes bsp role (58ms)
    ✔ Removes from bsps (115ms)
    ✔ Removes operator from validator ids (57ms)
    ✔ Should revert when trying to remove an operator that does not exist or has a different role (117ms)

  Tests all setters
    ✔ Lets Governance change the blockSpecimenRewardAllocation
    ✔ Emits BlockSpecimenRewardChanged
    ✔ Does not let non-governance change the blockSpecimenRewardAllocation
    ✔ Tests the getter for blockSpecimenRewardAllocation (43ms)
    ✔ Lets Governance change the blockSpecimenSessionDuration
    ✔ Emits SpecimenSessionDurationChanged
    ✔ Does not let non-governance change the blockSpecimenSessionDuration
    ✔ Tests the getter for blockSpecimenSessionDuration (49ms)
    ✔ Lets Governance change the blockSpecimenQuorum
    ✔ Emits SpecimenSessionQuorumChanged
    ✔ Does not let non-governance change the blockSpecimenQuorum
    ✔ Tests the getter for blockSpecimenQuorum (39ms)
    ✔ Lets a governance role set the staking manager address to a new address
    ✔ Emits StakingInterfaceChanged and successfully executes when governance calls
    ✔ Changes staking interface (916ms)
    ✔ Reverts when non-governance sets staking contract address to a new address
    ✔ Lets Governance change the minSubmissionsRequired
    ✔ Emits SpecimenSessionMinSubmissionChanged
    ✔ Does not let non-governance change the minSubmissionsRequired
    ✔ Tests the getter for minSubmissionsRequired (56ms)
    ✔ Lets Governance change the minSubmissionsRequired
    ✔ Emits NthBlockChanged
    ✔ Does not let non-governance change the nthBlock
    ✔ Tests the getter for nthBlock (50ms)
    ✔ Lets Governance change the setSecondsPerBlock
    ✔ Emits SecondsPerBlockCurrentChainChanged
    ✔ Does not let non-governance change the maxNumberOfHashesPer24H
    ✔ Tests the getter for maxNumberOfHashesPer24H (61ms)
    ✔ Lets Governance change the maxSubmissionsPerBlockHeight
    ✔ Emits BlockSpecimenMaxNumberOfHashesPer24HChanged
    ✔ Does not let non-governance change the maxSubmissionsPerBlockHeight (39ms)
    ✔ Tests the getter for maxSubmissionsPerBlockHeight (46ms)
    ✔ Tests the getter for maxSubmissionsPerBlockHeight (with revert) (39ms)
    ✔ Lets Governance change the chainSyncData and emits event with correct args
    ✔ Does not let non-governance change the chainSyncData (41ms)
    ✔ Reverts when seconds per block is 0 (39ms)
    ✔ Tests the getter for maxSubmissionsPerBlockHeight (41ms)
    ✔ Lets Governance change the allowedThreshold and emits event with correct args
    ✔ Does not let non-governance change the allowedThreshold
    ✔ Tests the getter for allowedThreshold
```

```
  Tests submitBlockSpecimenProof()
    ✔ Lets a BSP role submit a specimen proof (220ms)
    ✔ Reverts when a non-BSP submits a specimen proof (103ms)
    ✔ Reverts when invalid chain ID is provided (57ms)
    ✔ Emits BlockSpecimenProductionProofSubmitted event with correct args (69ms)
    ✔ Reverts when trying to submit out of bounds of live sync (256ms)
    ✔ Should revert when attempt to submit after session has closed (reached its deadline) (143ms)
    ✔ Should revert when attempt to submit after session has closed reached its deadline and being
finalized (188ms)
    ✔ Should revert when attempt to submit specimen hash for the same block height and block hash twice
(179ms)
    ✔ Should revert when attempt to submit when operator did not stake sufficiently when session has not
been started (117ms)
    ✔ Should revert when attempt to submit when operator did not stake sufficiently when session has
already been started (182ms)
    ✔ Does same block height on different chain IDs without collision (1320ms)
    ✔ Reverts when trying to submit for invalid block height (180ms)
    ✔ Reverts when trying to submit more than max number of submissions allowed per block height (281ms)
    ✔ Reverts when trying to submit for the same block hash per block height twice (217ms)

  Initialize contract
ProofChain upgraded to: 0xC073Eec654A6a30e221AD06591101413250Ac2dA
    ✔ works before and after upgrading (1356ms)


  101 passing (5m)


Version
=======
> solidity-coverage: v0.8.10

Instrumenting for coverage...
=============================

> BlockResultProofChain.sol
> BlockSpecimenProofChain.sol
> CovalentQueryTokenFaucet.sol
> ERC20Permit/ERC20Permit.sol
> ERC20Permit/IERC2612Permit.sol
> IOperationalStaking.sol
> OperationalStaking.sol

Compilation:
============

Nothing to compile

Network Info
============
> HardhatEVM: v2.19.5
> network:    hardhat



  Ownership
    ✔ Should return owner address same as signer. (864ms)
    ✔ Should access depositRewards, takeOutRewardTokens, setMaxCapMultiplier by owner. (795ms)
    ✔ Should not access depositRewards, takeOutRewardTokens, addValidator by not owner. (625ms)
    ✔ Should access rewardValidator, addValidator by proofChain. (679ms)
    ✔ Should not access rewardValidator, addValidator by not proofChain. (647ms)
    ✔ Should not access internal functions. (488ms)

  Account freezing
    ✔ Freezing an invalid validator should revert (490ms)
    ✔ Freezing a validator should emit ValidatorFrozen event (529ms)
    ✔ Freezing a validator twice should revert (574ms)
    ✔ Unfreezing a validator should emit ValidatorUnfrozen event (571ms)
    ✔ Unfreezing a not-frozen validator should revert (546ms)

  Add Validator
```

```
  ✔ Should change validators number. (957ms)
  ✔ Should emit event  with correct validator and commission rate. (494ms)
  ✔ Should add validator with correct commission rate. (932ms)
  ✔ Should add validator with correct address. (954ms)
  ✔ Should revert when validator address is 0. (511ms)
  ✔ Should revert when commission rate is 100% (558ms)

Deposit reward Tokens
  ✔ Should change balance of the contract and the owner. (342ms)
  ✔ Should change rewardPool. (605ms)
  ✔ Should emit RewardTokensDeposited event with correct amount. (565ms)
  ✔ Should revert with wrong inputs. (337ms)

Disable validator
  ✔ Should emit event with correct validator and disabled block. (714ms)
  ✔ Should return correct disabled block. (635ms)
  ✔ Should revert when trying to disable invalid validator. (621ms)

Enable validator
  ✔ Should be able to call stake after validator got enabled after being disabled. (870ms)
  ✔ Should emit event with correct validator and disabled block. (688ms)
  ✔ Should return correct disabled block. (661ms)
  ✔ Should revert when enabling invalid validator id. (711ms)
  ✔ Should revert when enabling under-staked validator (560ms)

Get all validators metadata
  ✔ Should return correct validator addresses (1086ms)
  ✔ Should return correct # of tokens staked (881ms)
  ✔ Should return correct # of tokens delegated (1087ms)
  ✔ Should return correct disabled at block number (1153ms)

Get delegator metadata
  ✔ Should return correct # of tokens staked by validator (981ms)
  ✔ Should return correct # of tokens staked by delegator (984ms)
  ✔ Should return correct amounts of unstakings (927ms)
  ✔ Should return correct end epochs of unstakings (1372ms)
  ✔ Should revert when validator id is invalid (689ms)

Get delegator total value locked
  ✔ Should return correct # of tokens locked with validator (979ms)
  ✔ Should return correct amounts with unstakings (908ms)

Get metadata
  ✔ Should return correct number of validators. (940ms)
  ✔ Should return correct CQT address. (156ms)
  ✔ Should return correct staking manager address. (485ms)
  ✔ Should return correct reward pool. (587ms)
  ✔ Should return correct delegator cool down . (159ms)
  ✔ Should return correct validator cool down . (157ms)
  ✔ Should return correct recover unstaking cool down . (159ms)
  ✔ Should return correct max cap multiplier. (164ms)
  ✔ Should return correct validator max stake. (161ms)

Get validator compounded staking data
  ✔ Should return correct # of tokens staked (868ms)
  ✔ Should return correct # of tokens delegated (882ms)

Get validator metadata
  ✔ Should return correct validator address (1086ms)
  ✔ Should return correct validator commission rate (1102ms)
  ✔ Should return correct # of tokens staked (851ms)
  ✔ Should return correct # of tokens delegated (1065ms)
  ✔ Should return correct disabled at block number (703ms)
  ✔ Should revert when validator id is invalid (664ms)

Get validators from start id to end id metadata
  ✔ Should return correct validator addresses (1265ms)
  ✔ Should return correct # of tokens staked (878ms)
  ✔ Should return correct # of tokens delegated (1079ms)
  ✔ Should return correct disabled at block number (761ms)
  ✔ Should revert with invalid end id (695ms)
  ✔ Should revert with invalid start and end ids (848ms)
```

```
Get validator staking data
  ✔ Should return correct # of tokens staked (897ms)
  ✔ Should return correct # of tokens delegated (887ms)
  ✔ Should revert when validator id is invalid (684ms)

Initialize contract
  ✔ Should emit Initialized event with correct args. (150ms)
  ✔ Cannot call initialize tx twice (172ms)

Pausability
  ✔ Pausing emits Paused event (518ms)
  ✔ Unpausing emits Unpaused event (495ms)
  ✔ Unpausing reverts when not paused (512ms)
  ✔ Validator should not be able to stake when contract is paused (650ms)
  ✔ Staking manager should still be able to perform admin functions when paused (542ms)

Recover Unstaking
  ✔ Should revert when recover invalid unstaking (864ms)
  ✔ Should revert when recover greater than staking (814ms)
  ✔ Should emit event when recovered unstake successfully (927ms)
  ✔ Should revert when try to recover the same unstake second time (895ms)
  ✔ Should not change contract balance (909ms)

  Redeem All Rewards
    ✔ Should emit redeem reward event with correct number of rewards when there are no delegators
(1075ms)
    ✔ Should emit redeem reward event with correct number of rewards when there are delegators (1269ms)
    ✔ Should change balances of the contract and delegator (960ms)
    ✔ Should revert with nothing to redeem (1346ms)
    ✔ Should revert with invalid beneficiary (710ms)

  Redeem Commission
    ✔ Should CommissionRewardRedeemed event with correct number of rewards when there are no delegators
(1155ms)
    ✔ Should emit CommissionRewardRedeemed event with correct number of rewards when there are delegators
(1206ms)
    ✔ Should change balances of the contract and delegator (1012ms)
    ✔ Should revert when validator is frozen (915ms)
    ✔ Should revert with nothing to redeem (1417ms)
    ✔ Should revert with invalid beneficiary when trying to redeem some commmission (715ms)
    ✔ Should revert with invalid beneficiary when trying to redeem all commmission (911ms)
    ✔ Should revert with invalid validator when trying to redeem (728ms)
    ✔ Should revert with invalid validator when trying to redeem (738ms)
    ✔ Should revert when delegator trying to redeem some commission (691ms)
    ✔ Should revert when delegator trying to redeem all commission (694ms)

  Redeem Rewards
    ✔ Should revert when requested amount 0 (695ms)
    ✔ Should revert when requested amount is too high (733ms)
    ✔ Should revert when trying to redeem from invalid validator (668ms)
    ✔ Should revert when validator is frozen (783ms)
    ✔ Should revert when redeem amount is too small (736ms)

Redelegate Unstaked
  ✔ Should redelegate partially and emit Redelegated and Staked events (1321ms)
  ✔ Should redelegate fully and emit event (1212ms)
  ✔ Should not be able to redelegate the same unstake fully twice (1483ms)
  ✔ Should change number of staked tokens under new validator (1246ms)
  ✔ Should revert when redelegating with enabled validator (1220ms)
  ✔ Should revert when validator is frozen (1176ms)
  ✔ Should revert when validators attempt to redelegate (1251ms)
  ✔ Should revert when redelegate greater than unstake (1239ms)
  ✔ Should revert when redelegating from invalid validator (1160ms)
  ✔ Should revert when redelegating invalid unstaking (1144ms)
  ✔ Should not change contract balance (1325ms)
  ✔ Should revert when redelegating from enabled validator that was disabled (1213ms)

Set renounce ownership
  ✔ Should change owner to zero address if owner is renounced. (507ms)

Reward validator
```

```
    ✔ Should change reward pool (989ms)
    ✔ Should commission available to redeem (785ms)
    ✔ Should emit Rewarded failed due to low pool event with correct validatorId and amount (776ms)
    ✔ Should emit Rewarded failed due to zero stake event with correct validatorId and amount  (635ms)
    ✔ Should emit Rewarded failed due to validator disabled event with correct validatorId and amount
(615ms)
    ✔ Should emit Rewarded failed due to validator frozen event with correct validatorId and amount
(611ms)
    ✔ Should revert when given uneven number of ids and reward amounts (695ms)
    ✔ get validators compounded staking data (753ms)

  Set max cap multiplier
    ✔ Should change max cap multiplier. (623ms)
    ✔ Should emit StakingManagerChanged event with correct address. (585ms)
    ✔ Should be able to delegate more if multiplier increases and should revert when attempted to
delegate above max cap. (1157ms)
    ✔ Should revert if set to 0. (740ms)

  Set staking manager address
    ✔ Should change staking manager address. (529ms)
    ✔ Should emit StakingManagerChanged event with correct address. (491ms)
    ✔ Should revert when set to zero address. (495ms)

  Set validator address
    ✔ Should change staking validator address. (680ms)
    ✔ Should transfer rewards. (835ms)
    ✔ Should transfer stakings. (770ms)
    ✔ Should transfer unstakings. (901ms)
    ✔ Should emit ValidatorAddressChanged event with correct address. (848ms)
    ✔ Should not access setValidatorAddress by not validator. (754ms)
    ✔ Should revert when the new address is 0. (672ms)
    ✔ Should revert when the new address is the old one. (642ms)
    ✔ Should revert when the validator id is invalid. (655ms)
    ✔ Should revert when validator is frozen (727ms)
    ✔ Should revert when trying to set address to a delegator. (1122ms)
    ✔ Should transfer unstakings when the max amount is used. (18005ms)
    ✔ Should revert when more than the max amount of unstakings is used. (16682ms)

  Set validator commission rate
    ✔ Should change validator commission rate. (837ms)
    ✔ Should emit ValidatorCommissionRateChanged event with correct validator id and amount. (820ms)
    ✔ Should emit correct amount of validator commission rewards. (1015ms)
    ✔ Should emit correct amount of delegator rewards. (964ms)
    ✔ Should revert with invalid validator id. (692ms)
    ✔ Should revert if set to >= 10^18. (1099ms)

  Set validator max stake
    ✔ Should change validator max stake amount. (560ms)
    ✔ Should emit ValidatorMaxCapChanged event with correct amount. (519ms)
    ✔ Should revert when max stake is set to 0. (692ms)

  Staking
    ✔ Should stake when validator is disabled (721ms)
    ✔ Should revert when validator is frozen (723ms)
    ✔ Should revert when transfer not approved (704ms)
    ✔ Should stake 1 token and emit event with correct number (835ms)
    ✔ Should return correct delegated #  (1524ms)
    ✔ Should revert when stake by validator is more than stake max cap (757ms)
    ✔ Should revert when stake to invalid validator (674ms)
    ✔ Should change contract balance (669ms)
    ✔ Should change delegator balance (645ms)
    ✔ Should succeed when stake by validator is at max cap (669ms)
    ✔ Should revert when stake amount is too small (635ms)
    ✔ Should revert when staking would result in a position below minDelegatorStake (635ms)

  Take out reward Tokens
    ✔ Should change balance of the contract and the owner. (350ms)
    ✔ Should revert with wrong inputs. (533ms)
    ✔ Should change rewardPool. (667ms)
    ✔ Should emit AllocatedTokensTaken event with correct amount. (645ms)
    ✔ Should revert when reward pool is too small. (488ms)
```

```
  Transfer Unstaked
    ✔ Should transfer out after cool down ends, delegator (1097ms)
    ✔ Should transfer out after cool down ends, validator (1892ms)
    ✔ Should transfer out partially (1356ms)
    ✔ Should change balance of the contract and the owner. (1270ms)
    ✔ Should transfer out after cool down ends, validator (2385ms)
    ✔ Should revert with wrong unstaking id (1092ms)
    ✔ Should revert when the transfer amount is higher than unstaked (1079ms)
    ✔ Should revert when trying to attempt transfer the same unstake twice (909ms)
    ✔ Should revert when cool down did not end, delegator (1031ms)
    ✔ Should revert when cool down did not end, validator (1019ms)
    ✔ Should revert when given invalid validator id (1028ms)
    ✔ Should allow immediate unstaking when delegator cooldown is set to 0 (750ms)
    ✔ Should allow immediate unstaking when validator cooldown is set to 0 (647ms)
    ✔ Should allow updated unstaking period, while previous unstaking periods remain the same (when
delegator cooldown is changed) (1102ms)
    ✔ Should allow updated unstaking period, while previous unstaking periods remain the same (when
validator cooldown is changed) (12077ms)

  Underflow protection
    ✔ Redeeming rewards should revert on underflow (943ms)
    ✔ Unstaking should revert on underflow (968ms)

  Unstaking
    ✔ Should revert when unstake is more than staked (848ms)
    ✔ Should revert when validator is frozen (798ms)
    ✔ Should revert when unstake amount is too small (774ms)
    ✔ Should revert when unstake would result in a position below minDelegatorStake (923ms)
    ✔ Should disable validator when unstake results in a position below minValidatorEnableStake (785ms)
    ✔ Should revert when unstake beyond max cap (840ms)
    ✔ Should unstake with safe max cap (833ms)
    ✔ Should unstake all, leaving zero remaining stake (780ms)
    ✔ Should revert when unstaking all without stake (702ms)
    ✔ Should unstake beyond max cap when validator is disabled (864ms)
    ✔ Should emit event when unstaked successfully (750ms)
    ✔ Should not change balance of contract or delegator (798ms)
    ✔ Should revert when validator is invalid (779ms)


  196 passing (3m)
```

Code Coverage

Both `BlockSpecimenProofChain` and `OperationalStaking` has good test coverage > 90%. However we noticed some failed tests despite the high coverage. The team should take a look and fix those failed tests.

Update: After the fix review, we re-run the unit tests in the commit `1ea6c5` :

- `npx hardhat coverage --testfiles test/cqt-staking/unit-tests`
- `npx hardhat coverage --testfiles test/block-specimen-proof-chain/unit-tests`

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	43.84	44.12	40.15	45.56	
IOperationalStaking.sol	100	100	100	100	
OperationalStaking.sol	99.26	82.82	100	99.17	539,971,980

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	33	23.37	35.61	33.76	
BlockSpecimenProofChain.	95.83	82.14	92.11	96.68	...

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
sol					604,605,606

Changelog

- 2024-01-31 - Initial report
- 2024-03-06 - Fix review report
- 2024-03-13 - Second fix review report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp