Quantstamp Security Assessment Certificate

January 16th 2023 — Quantstamp Verified

Covalent

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

Executive Summary

Type	Staking Program
Auditors	Danny Aksenov, Security Auditor Jennifer Wu, Auditing Engineer Hytham Farah, Auditing Engineer II Sina Pilehchiha, Audit Engineer I
Timeline	2022-12-12 through 2022-12-12
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Nomad Bridge Hack Recovery Plan Protocol Summary
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
covalenthq/bsp-staking	0ecfaaa initial audit
covalenthq/bsp-staking	8cd5087 fixes

Total Issues	**6** (3 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**2** (1 Resolved)
Informational Risk Issues	**4** (2 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	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.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
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).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The goal of this audit was to evaluate the Covalent team's plan to replace the old CQT token with a new CQT token, and zero out the stakes, rewards, and unstakes of a predefined list of delegators who will not receive compensation. To accomplish this, the plan is to upgrade the `OperationalStaking` contract to the `MigrationOperationalStaking` contract, set the stakes, rewards, and unstakes of the specified delegators to zero, withdraw all CQT to a recovery wallet, change the CQT address, and transfer the new CQT to the contract using the `MadCQTBurnt` event to calculate the difference between the old CQT balance and the sum of `MadCQTBurnt`. Finally, the `MigrationOperationalStaking` contract will be upgraded back to the `OperationalStaking` contract. For this audit, Quantstamp primarily focused on the `MigrationOperationalStaking` contract and evaluated the tests written for this flow. Overall we found the code to be well-written, well-documented, and following best practices. We have not found any significant security vulnerabilities. To maximize certainty in this process, we recommend updating the tests to cover all steps of the outlined flow and adding some additional checks.

ID	Description	Severity	Status
QSP-1	Upgradeable Misconfigurations	Informational	Acknowledged
QSP-2	Testing Can Be More Extensive	Low	Fixed
QSP-3	`CQT` Token Address Can Be Changed Before Withdrawing All `MadCQT` Tokens	Low	Acknowledged
QSP-4	Event Omitted	Informational	Fixed
QSP-5	Missing Input Validation	Informational	Fixed
QSP-6	Upgradability	Informational	Acknowledged

Quantstamp Audit Breakdown

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

DISCLAIMER:
It is important to note that `OperationalStaking.sol` was evaluated for issues in regards to the flow outlined in the summary.
In addition, if the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features 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

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. 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:
   i. 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.
   ii. 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, clarify, 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.

Toolset

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

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`
3. Run `slither-check-upgradeability . contracts/OperationalStaking.sol OperationalStaking --new-contract-name MigrationOperationalStaking`

Findings

QSP-1 Upgradeable Misconfigurations

Severity: Informational

Status: Acknowledged

File(s) affected: `OperationalStaking.sol`

Description: The implementation contracts are not guaranteed to have their initializer disabled. This makes it possible for a malicious actor to call the `initialize()` function on the implementation contracts.

Recommendation: Add `_disableInitializers()` to the constructor of `OperationalStaking.sol`.

Update: Initially a medium-severity issue, we have downgraded this issue to informational level in concurrence with the Covalent team's assessment of this issue posing any serious security risks.

QSP-2 Testing Can Be More Extensive

Severity: Low Risk

Status: Fixed

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

Description: While the current test implementation has great coverage, the following items are missing from the tests:

- Address `0x1DB596c09f5B37013B3cc263B9903D2474050F3f`, which was recently added in the following commit.
- The following event: "Transfer the new CQT to the contract using MadCQTBurnt event to calculate the difference: how much oldCQT balance was before minus sum of MadCQTBurnt".

Recommendation: Consider implementing the above changes for better test coverage.

Update: The suggested changes to the tests were made in the following commit.

QSP-3 `CQT` Token Address Can Be Changed Before Withdrawing All `MadCQT` Tokens

Severity: Low Risk

Status: Acknowledged

File(s) affected: `MigrationOperationalStaking.sol`

Description: The `CQT` token address can be upgraded before `withdrawAllMadCQT()` is invoked.

Recommendation: Before setting the `CQT` address in `setCQTAddress()` verify the `CQT` balance of the contract is zero.

Update: The Covalent team plans on verifying that the token balance has been zero'ed out prior to changing the address off-chain.

QSP-4 Event Omitted

Severity: Informational

Status: Fixed

File(s) affected: `MigrationOperationalStaking.sol`

Description: The following function is missing an event:

- `MigrationOperationalStaking.withdrawAllMadCQT()`

Recommendation: Consider emitting the `amount` transferred and `recovery address` when `withdrawAllMadCQT()` is invoked.

Update: The suggested change was implemented in the following commit.

QSP-5 Missing Input Validation

Severity: Informational

Status: Fixed

File(s) affected: `MigrationOperationalStaking.sol`

Description: Currently, `setCQTAddress()` only checks that the provided address is non-zero, however, while highly unlikely, there is a possibility to set the new CQT address to original CQT address. Since this a critical operation, it might be beneficial to maximize certainty.

Recommendation: Consider validating that the `newCQT` address is not equal to the current `CQT` address.

Update: The suggested change was implemented in the following commit.

QSP-6 Upgradability

Severity: Informational

Status: Acknowledged

File(s) affected: `MigrationOperationalStaking.sol`

Description: While upgradability is not a vulnerability in itself, users should be aware that the Operational Staking contract can be upgraded at any given time. This audit does not guarantee the behavior of future upgraded contracts.

Recommendation: The fact that the contract can be upgraded and reasons for future upgrades should be communicated to users prior to upgrading.

Automated Analyses

Slither

Slither has produced several findings, but we have confirmed them to be false positives.

```
INFO:Slither:
OperationalStaking (contracts/OperationalStaking.sol#7-684) needs to be initialized by OperationalStaking.initialize(address,uint128,uint128,uint128,uint128) (contracts/OperationalStaking.sol#95-109).
Reference: https://github.com/crytic/slither/wiki/Upgradeability-Checks#initialize-function
INFO:Slither:
Different variables between OperationalStaking (contracts/OperationalStaking.sol#7-684) and MigrationOperationalStaking (contracts/MigrationOperationalStaking.sol#7-260)
         OperationalStaking._validators (contracts/OperationalStaking.sol#21)
         MigrationOperationalStaking._validators (contracts/MigrationOperationalStaking.sol#21)
Reference: https://github.com/crytic/slither/wiki/Upgradeability-Checks#incorrect-variables-with-the-v2
INFO:Slither:2 findings, 21 detectors run

MigrationOperationalStaking.rewardPool (contracts/MigrationOperationalStaking.sol#14) is never initialized. It is used in:
        - MigrationOperationalStaking.getMetadata() (contracts/MigrationOperationalStaking.sol#187-202)
MigrationOperationalStaking.validatorCoolDown (contracts/MigrationOperationalStaking.sol#15) is never initialized. It is used in:
        - MigrationOperationalStaking.getMetadata() (contracts/MigrationOperationalStaking.sol#187-202)
MigrationOperationalStaking.delegatorCoolDown (contracts/MigrationOperationalStaking.sol#16) is never initialized. It is used in:
        - MigrationOperationalStaking.getMetadata() (contracts/MigrationOperationalStaking.sol#187-202)
MigrationOperationalStaking.maxCapMultiplier (contracts/MigrationOperationalStaking.sol#17) is never initialized. It is used in:
        - MigrationOperationalStaking.getMetadata() (contracts/MigrationOperationalStaking.sol#187-202)
MigrationOperationalStaking.validatorMaxStake (contracts/MigrationOperationalStaking.sol#18) is never initialized. It is used in:
        - MigrationOperationalStaking.getMetadata() (contracts/MigrationOperationalStaking.sol#187-202)
MigrationOperationalStaking.stakingManager (contracts/MigrationOperationalStaking.sol#19) is never initialized. It is used in:
        - MigrationOperationalStaking.getMetadata() (contracts/MigrationOperationalStaking.sol#187-202)
MigrationOperationalStaking.validatorsN (contracts/MigrationOperationalStaking.sol#20) is never initialized. It is used in:
        - MigrationOperationalStaking.burnDefaultDelegators() (contracts/MigrationOperationalStaking.sol#107-124)
        - MigrationOperationalStaking.getMetadata() (contracts/MigrationOperationalStaking.sol#187-202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

ProofChain._rewardParticipants(ProofChain.BlockSpecimenSession,uint64,uint64,bytes32,bytes32) (contracts/ProofChain.sol#492-524) deletes ProofChain.BlockHash (contracts/ProofChain.sol#48-51) which contains a mapping:
        -delete session.blockHashes[blockHash] (contracts/ProofChain.sol#523)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#deletion-on-mapping-containing-a-structure

Reentrancy in ProofChain.finalizeAndRewardSpecimenSession(uint64,uint64) (contracts/ProofChain.sol#434-474):
        External calls:
        - _rewardParticipants(session,chainId,blockHeight,agreedBlockHash,agreedSpecimenHash) (contracts/ProofChain.sol#468)
                - _stakingInterface.rewardValidators(ids,rewards) (contracts/ProofChain.sol#520)
        State variables written after the call(s):
        - session.requiresAudit = true (contracts/ProofChain.sol#471)
        - session.sessionDeadline = 0 (contracts/ProofChain.sol#473)
Reentrancy in ProofChain.removeBSPOperator(address) (contracts/ProofChain.sol#211-218):
        External calls:
        - _removeBSPOperatorFromActiveInstances(operator) (contracts/ProofChain.sol#213)
                - _stakingInterface.disableValidator(validatorId,block.number) (contracts/ProofChain.sol#169)
        State variables written after the call(s):
        - operatorRoles[operator] = 0 (contracts/ProofChain.sol#216)
        - validatorIDs[operator] = 0 (contracts/ProofChain.sol#215)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

ProofChain.finalizeAndRewardSpecimenSession(uint64,uint64).agreedBlockHash (contracts/ProofChain.sol#445) is a local variable never initialized
ProofChain.submitBlockSpecimenProof(uint64,uint64,bytes32,bytes32,string).baseStake_scope_1 (contracts/ProofChain.sol#411) is a local variable never initialized
ProofChain.finalizeAndRewardSpecimenSession(uint64,uint64).agreedSpecimenHash (contracts/ProofChain.sol#446) is a local variable never initialized
ProofChain.finalizeAndRewardSpecimenSession(uint64,uint64).contributorsN (contracts/ProofChain.sol#441) is a local variable never initialized
ProofChain.finalizeAndRewardSpecimenSession(uint64,uint64).max (contracts/ProofChain.sol#444) is a local variable never initialized
ProofChain.submitBlockSpecimenProof(uint64,uint64,bytes32,bytes32,string).delegateStakes_scope_2 (contracts/ProofChain.sol#411) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ProofChain.initialize(address,address) (contracts/ProofChain.sol#125-144) ignores return value by _governors.add(msg.sender) (contracts/ProofChain.sol#128)
ProofChain.initialize(address,address) (contracts/ProofChain.sol#125-144) ignores return value by _roleNames.add(GOVERNANCE_ROLE) (contracts/ProofChain.sol#130)
ProofChain.initialize(address,address) (contracts/ProofChain.sol#125-144) ignores return value by _roleNames.add(BLOCK_SPECIMEN_PRODUCER_ROLE) (contracts/ProofChain.sol#131)
ProofChain.initialize(address,address) (contracts/ProofChain.sol#125-144) ignores return value by _roleNames.add(AUDITOR_ROLE) (contracts/ProofChain.sol#132)
ProofChain.initialize(address,address) (contracts/ProofChain.sol#125-144) ignores return value by _governors.remove(msg.sender) (contracts/ProofChain.sol#139)
ProofChain.initialize(address,address) (contracts/ProofChain.sol#125-144) ignores return value by _governors.add(initialOwner) (contracts/ProofChain.sol#142)
ProofChain.addValidator(address,uint128) (contracts/ProofChain.sol#149-151) ignores return value by _stakingInterface.addValidator(validator,commissionRate) (contracts/ProofChain.sol#150)
ProofChain._removeBSPOperatorFromActiveInstances(address) (contracts/ProofChain.sol#164-170) ignores return value by _blockSpecimenProducers.remove(operator) (contracts/ProofChain.sol#165)
ProofChain.enableBSPOperator(address) (contracts/ProofChain.sol#175-184) ignores return value by _blockSpecimenProducers.add(operator) (contracts/ProofChain.sol#179)
ProofChain.addBSPOperator(address,uint128) (contracts/ProofChain.sol#200-206) ignores return value by _validatorOperators[validatorId].add(operator) (contracts/ProofChain.sol#204)
ProofChain.removeBSPOperator(address) (contracts/ProofChain.sol#211-218) ignores return value by _validatorOperators[validatorIDs[operator]].remove(operator) (contracts/ProofChain.sol#214)
ProofChain.addAuditor(address) (contracts/ProofChain.sol#223-228) ignores return value by _auditors.add(auditor) (contracts/ProofChain.sol#226)
ProofChain.removeAuditor(address) (contracts/ProofChain.sol#233-238) ignores return value by _auditors.remove(auditor) (contracts/ProofChain.sol#236)
ProofChain.addGovernor(address) (contracts/ProofChain.sol#243-248) ignores return value by _governors.add(governor) (contracts/ProofChain.sol#246)
ProofChain.removeGovernor(address) (contracts/ProofChain.sol#253-258) ignores return value by _governors.remove(governor) (contracts/ProofChain.sol#256)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

CovalentQueryTokenFaucet.constructor(string,string,uint256).name (contracts/CovalentQueryTokenFaucet.sol#17) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64) (function)
        - IERC20Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#17) (function)
CovalentQueryTokenFaucet.constructor(string,string,uint256).symbol (contracts/CovalentQueryTokenFaucet.sol#18) shadows:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72) (function)
        - IERC20Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#22) (function)
CovalentQueryTokenFaucet.constructor(string,string,uint256).totalSupply (contracts/CovalentQueryTokenFaucet.sol#19) shadows:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96) (function)
        - IERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#27) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Variable 'ProofChain.submitBlockSpecimenProof(uint64,uint64,bytes32,bytes32,string).delegateStakes (contracts/ProofChain.sol#385)' in ProofChain.submitBlockSpecimenProof(uint64,uint64,bytes32,bytes32,string)
(contracts/ProofChain.sol#360-429) potentially used before declaration: (baseStake,delegateStakes) = _stakingInterface.getValidatorCompoundedStakingData(validatorIDs[msg.sender]) (contracts/ProofChain.sol#411)
Variable 'ProofChain.submitBlockSpecimenProof(uint64,uint64,bytes32,bytes32,string).baseStake (contracts/ProofChain.sol#385)' in ProofChain.submitBlockSpecimenProof(uint64,uint64,bytes32,bytes32,string)
(contracts/ProofChain.sol#360-429) potentially used before declaration: (baseStake,delegateStakes) = _stakingInterface.getValidatorCompoundedStakingData(validatorIDs[msg.sender]) (contracts/ProofChain.sol#411)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in ProofChain._rewardParticipants(ProofChain.BlockSpecimenSession,uint64,uint64,bytes32,bytes32) (contracts/ProofChain.sol#492-524):
        External calls:
        - _stakingInterface.rewardValidators(ids,rewards) (contracts/ProofChain.sol#520)
        Event emitted after the call(s):
        - BlockSpecimenRewardAwarded(chainId,blockHeight,blockHash,specimenHash) (contracts/ProofChain.sol#521)
Reentrancy in OperationalStaking._stake(uint128,uint128,bool) (contracts/OperationalStaking.sol#282-318):
        External calls:
        - _transferToContract(msg.sender,amount) (contracts/OperationalStaking.sol#316)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#93)
                - CQT.safeTransferFrom(from,address(this),amount) (contracts/OperationalStaking.sol#247)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
        External calls sending eth:
```

```
        - _transferToContract(msg.sender,amount) (contracts/OperationalStaking.sol#316)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
        Event emitted after the call(s):
        - Staked(validatorId,msg.sender,amount) (contracts/OperationalStaking.sol#317)
Reentrancy in OperationalStaking.depositRewardTokens(uint128) (contracts/OperationalStaking.sol#120-127):
        External calls:
        - _transferToContract(msg.sender,amount) (contracts/OperationalStaking.sol#125)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#93)
                - CQT.safeTransferFrom(from,address(this),amount) (contracts/OperationalStaking.sol#247)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
        External calls sending eth:
        - _transferToContract(msg.sender,amount) (contracts/OperationalStaking.sol#125)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
        Event emitted after the call(s):
        - RewardTokensDeposited(amount) (contracts/OperationalStaking.sol#126)
Reentrancy in ProofChain.disableBSPOperator(address) (contracts/ProofChain.sol#190-195):
        External calls:
        - _removeBSPOperatorFromActiveInstances(operator) (contracts/ProofChain.sol#193)
                - _stakingInterface.disableValidator(validatorId,block.number) (contracts/ProofChain.sol#169)
        Event emitted after the call(s):
        - OperatorDisabled(operator) (contracts/ProofChain.sol#194)
Reentrancy in ProofChain.enableBSPOperator(address) (contracts/ProofChain.sol#175-184):
        External calls:
        - _stakingInterface.enableValidator(validatorId) (contracts/ProofChain.sol#182)
        Event emitted after the call(s):
        - OperatorEnabled(operator) (contracts/ProofChain.sol#183)
Reentrancy in OperationalStaking.redeemCommission(uint128,address,uint128) (contracts/OperationalStaking.sol#464-482):
        External calls:
        - _transferFromContract(beneficiary,amount) (contracts/OperationalStaking.sol#480)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#93)
                - CQT.safeTransfer(to,amount) (contracts/OperationalStaking.sol#254)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
        External calls sending eth:
        - _transferFromContract(beneficiary,amount) (contracts/OperationalStaking.sol#480)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
        Event emitted after the call(s):
        - CommissionRewardRedeemed(validatorId,beneficiary,amount) (contracts/OperationalStaking.sol#481)
Reentrancy in ProofChain.removeBSPOperator(address) (contracts/ProofChain.sol#211-218):
        External calls:
        - _removeBSPOperatorFromActiveInstances(operator) (contracts/ProofChain.sol#213)
                - _stakingInterface.disableValidator(validatorId,block.number) (contracts/ProofChain.sol#169)
        Event emitted after the call(s):
        - OperatorRemoved(operator) (contracts/ProofChain.sol#217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/ERC20Permit/ERC20Permit.sol#53-73) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,CovalentPermit: expired deadline) (contracts/ERC20Permit/ERC20Permit.sol#62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ERC20Permit.constructor() (contracts/ERC20Permit/ERC20Permit.sol#24-39) uses assembly
        - INLINE ASM (contracts/ERC20Permit/ERC20Permit.sol#26-28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#186-189)
EnumerableSetUpgradeable.values(EnumerableSetUpgradeable.AddressSet) (node_modules/@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol#274-283) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol#278-280)
EnumerableSetUpgradeable.values(EnumerableSetUpgradeable.UintSet) (node_modules/@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol#347-356) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol#351-353)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.4']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#4)
        - ^0.8.4 (contracts/CovalentQueryTokenFaucet.sol#1)
        - ^0.8.4 (contracts/ERC20Permit/ERC20Permit.sol#1)
        - ^0.8.4 (contracts/ERC20Permit/IERC2612Permit.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Different versions of Solidity are used:
        - Version used: ['0.8.13', '^0.8.0', '^0.8.1', '^0.8.13']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol#4)
        - ^0.8.13 (contracts/IOperationalStaking.sol#2)
        - 0.8.13 (contracts/MigrationOperationalStaking.sol#2)
        - 0.8.13 (contracts/OperationalStaking.sol#2)
        - 0.8.13 (contracts/ProofChain.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#4) allows old versions
Pragma version^0.8.4 (contracts/CovalentQueryTokenFaucet.sol#1) allows old versions
Pragma version^0.8.4 (contracts/ERC20Permit/ERC20Permit.sol#1) allows old versions
Pragma version^0.8.4 (contracts/ERC20Permit/IERC2612Permit.sol#1) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol#4) allows old versions
Pragma version^0.8.13 (contracts/IOperationalStaking.sol#2) allows old versions
Pragma version0.8.13 (contracts/MigrationOperationalStaking.sol#2) allows old versions
Pragma version0.8.13 (contracts/OperationalStaking.sol#2) allows old versions
Pragma version0.8.13 (contracts/ProofChain.sol#2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

OperationalStaking (contracts/OperationalStaking.sol#7-684) should inherit from IOperationalStaking (contracts/IOperationalStaking.sol#4-26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance

Variable ERC20Permit.DOMAIN_SEPARATOR (contracts/ERC20Permit/ERC20Permit.sol#22) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Function OwnableUpgradeable.__Ownable_init() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#29-31) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#33-35) is not in mixedCase
Variable OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#87) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Variable MigrationOperationalStaking.CQT (contracts/MigrationOperationalStaking.sol#13) is not in mixedCase
Variable MigrationOperationalStaking._validators (contracts/MigrationOperationalStaking.sol#21) is not in mixedCase
Variable OperationalStaking.CQT (contracts/OperationalStaking.sol#13) is not in mixedCase
```

```
Variable OperationalStaking._validators (contracts/OperationalStaking.sol#21) is not in mixedCase
Variable ProofChain._stakingInterface (contracts/ProofChain.sol#12) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable MigrationOperationalStaking._validators (contracts/MigrationOperationalStaking.sol#21) is too similar to MigrationOperationalStaking.validatorsN (contracts/MigrationOperationalStaking.sol#20)
Variable OperationalStaking._validators (contracts/OperationalStaking.sol#21) is too similar to OperationalStaking.validatorsN (contracts/OperationalStaking.sol#20)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#87) is never used in MigrationOperationalStaking (contracts/MigrationOperationalStaking.sol#7-260)
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#87) is never used in OperationalStaking (contracts/OperationalStaking.sol#7-684)
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#87) is never used in ProofChain (contracts/ProofChain.sol#8-599)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

MigrationOperationalStaking.delegatorCoolDown (contracts/MigrationOperationalStaking.sol#16) should be constant
MigrationOperationalStaking.maxCapMultiplier (contracts/MigrationOperationalStaking.sol#17) should be constant
MigrationOperationalStaking.rewardPool (contracts/MigrationOperationalStaking.sol#14) should be constant
MigrationOperationalStaking.stakingManager (contracts/MigrationOperationalStaking.sol#19) should be constant
MigrationOperationalStaking.validatorCoolDown (contracts/MigrationOperationalStaking.sol#15) should be constant
MigrationOperationalStaking.validatorMaxStake (contracts/MigrationOperationalStaking.sol#18) should be constant
MigrationOperationalStaking.validatorsN (contracts/MigrationOperationalStaking.sol#20) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

Adherence to Best Practices

- Consider combining the `DelegatorBurnt` and `MadCQTBurnt` event together for improved readability.
- Some gas optimizations can be made on lines `#206, #214, #238, #239`, to store variables pulled out of `storage` in `memory`, as no state changes are being made in these functions.
- Consider implementing custom errors instead of `require()` statements. They allow for more detailed information to be returned in the form of variables, they permit more fine-grained error handling based on the error data, and they tend to be more gas efficient than require statements.

Test Results

Test Suite Results

All tests are passing.

```
 Initialize contract
Operational Staking upgraded to: 0xfdF85083AD86Fc0846c7812Fd39F5b0E1e74D321
Operational Staking upgraded to: 0xfdF85083AD86Fc0846c7812Fd39F5b0E1e74D321
     ✓ Should upgrade to migration contract and back to original
     ✓ Should emit MadCQTWithdrawn CQT address in between
Deploying CQT Faucet with the account: 0x8D1f2eBFACCf1136dB76FDD1b86f1deDE2D23852
     ✓ Should change CQT address in between
     ✓ Should change virtual CQT balances of default accounts
     ✓ Old balance minus sum of amounts in MadCQTBurnt events should match the total staked and delegated
     ✓ Should change virtual CQT balances of given accounts
     ✓ Should change CQT balance of the contract to 0 and increase the recipient balance to the correct value
     ✓ Should not access withdrawMadCQT, burnDefaultBalances, setCQTAddress, renounceOwnership by not owner.
     ✓ Should revert when the given CQT address is zero
     ✓ Should revert when attempt to set CQT address with balance > 0
     ✓ Should revert when the given CQT address is zero
     ✓ Should revert when the given CQT address is the same as before
     ✓ Should not burn self staked tokens.
     ✓ Should revert when providing invalid validator id.
     ✓ Should return correct # of tokens staked
     ✓ Should return correct # of tokens delegated
     ✓ Should not change the owner if owner is renounced.
     ✓ Should return correct # of tokens staked by validator
     ✓ Should return correct # of tokens staked by delegator
     ✓ Should return correct amounts of unstakings
     ✓ Should return correct end epochs of unstakings
     ✓ Should revert when validator id is invalid
     ✓ Should return correct # of tokens staked
     ✓ Should return correct # of tokens delegated
     ✓ Should revert when validator id is invalid
```

Code Coverage

The test suite has high branch coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
MigrationOperationalStaking.sol	96.43	93.75	100	98.15	159

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

`5bbc08b56d51531337b113145ac2928e428baf0250b70e63fd884f0d4c7878c1 ./contracts/OperationalStaking.sol`

`6dfda79bedcf569a277782a68149984fd5269f11e76ab2f7c3f9406c9e1413ff ./contracts/MigrationOperationalStaking.sol`

Tests

`0da5750fdd2caa3f9f8817dc188548da90491960aced8607df8e7d1b60491aee ./nomad-hack-recovery-migration/upgrades.js`

Changelog

- 2022-12-17 - Initial Report
- 2023-01-16 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors 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 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

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, Aave, 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