Quantstamp Security Assessment Certificate

QUANTSTAMP VERIFIED SECURITY CERTIFICATE

January 21st 2021 — Quantstamp Verified

Barn Bridge

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	Staking and Governance Contracts
Auditors	Ed Zulkoski, Senior Security Engineer Leonardo Passos, Senior Research Engineer Poming Lee, Research Engineer
Timeline	2021-01-11 through 2021-01-21
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Online Documentation
Documentation Quality	Medium
Test Quality	High

Source Code

Repository	Commit
BarnBridge-DAO	3238002
BarnBridge-Barn	3a0f8de

Goals

- Can funds be locked or stolen?
- Is the diamond pattern used correctly?

Total Issues	**13**	(0 Resolved)
High Risk Issues	0	(0 Resolved)
Medium Risk Issues	0	(0 Resolved)
Low Risk Issues	**5**	(0 Resolved)
Informational Risk Issues	**6**	(0 Resolved)
Undetermined Risk Issues	**2**	(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).
Resolved	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

During the audit, a number of issues were uncovered of varying severity. No high severity issues were found, however certain functions require further documentation in order to assess their correctness, as noted in several issues below. We recommend improving the inline documentation as well as expanding the specification document, as well as addressing all findings before using the code in production.

ID	Description	Severity	Status
QSP-1	Multiple copies of a transaction can be queued at the same time	Low	Unresolved
QSP-2	`_proposalCancelledViaCounterProposal` does not use the `bondStakedAtTs`	Low	Unresolved
QSP-3	Ownership can be renounced	Low	Unresolved
QSP-4	Unchecked `transferFrom` return value	Low	Unresolved
QSP-5	Functions do not check if contract is initialized	Low	Unresolved
QSP-6	Unlocked Pragma	Informational	Unresolved
QSP-7	Unchecked function arguments	Informational	Unresolved
QSP-8	Unchecked return value from `executeTransaction`	Informational	Unresolved
QSP-9	`setupPullToken` should not be invoked multiple times	Informational	Unresolved
QSP-10	Users without voting power are still flagged as having voted	Informational	Unresolved
QSP-11	Privileged Roles and Ownership	Informational	Unresolved
QSP-12	Unclear use-case for `receive` function in `Barn.sol`	Undetermined	Unresolved
QSP-13	Unclear timestamp setup logic in `propose`	Undetermined	Unresolved

Quantstamp Audit Breakdown

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

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 v0.7.0

- Mythril v0.22.16

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`

Findings

QSP-1 Multiple copies of a transaction can be queued at the same time

Severity: Low Risk

Status: Unresolved

File(s) affected: `Governance.sol`

Description: There is an implicit requirement that a transaction cannot be duplicated in the queue as per L174-177: `require(!queuedTransactions[_getTxHash(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], eta)], ...)`. This can be circumvented in the following way.

1. Create two proposals `P_1` and `P_2` that contain the same transaction `T`. (This must be done with multiple accounts to avoid the "One live proposal per proposer" check on L143.) Assume both proposals have been accepted, but not yet queued.
2. Queue `P_1`.
3. Create a third proposal `P_3` that contains `T` and, as the proposal creator, immediately cancel it. This sets `queuedTransactions[T] = false` via `Bridge.cancelTransaction`.
4. Queue `P_2`, which is now allowed since the check on L174-177 passes.

Recommendation: Clarify if this scenario is allowable. Revise the queued transaction logic if necessary.

QSP-2 `_proposalCancelledViaCounterProposal` does not use the `bondStakedAtTs`

Severity: Low Risk

Status: Unresolved

File(s) affected: `Governance.sol`

Description: When voting on a cancellation, vote weights are determined by `votingPowerAtTs`. The function `_proposalCancelledViaCounterProposal` should analogously use `bondStakedAtTs` to determine the ratio of affirmative cancellation votes.

Recommendation: Change the function to use `bondStakedAtTs`.

QSP-3 Ownership can be renounced

Severity: Low Risk

Status: Unresolved

File(s) affected: `LibOwnership.sol`

Description: In `setContractOwner`, a target owner can currently be set to `address(0)`, which effectively allows an owner to renounce his ownership. In the latter case, any privileged operation shall never be executed upon renouncing one's ownership.

Recommendation: Confirm if an owner could renounce his ownership; if not, make sure the new owner is not `address(0)`.

QSP-4 Unchecked `transferFrom` return value

Severity: Low Risk

Status: Unresolved

File(s) affected: `BarnFacet.sol`

Description: In the `deposit` and `withdraw` functions, the `transferFrom` return value is not checked, which is unadvisable. If `transferFrom` returns false, it would flag an error that currently would be missed by the platform, leaving users uninformed.

Recommendation: Wrap the `transferFrom` function in a require statement, adding a corresponding error message in case the transfer returns false.

QSP-5 Functions do not check if contract is initialized

Severity: Low Risk

Status: Unresolved

File(s) affected: `BarnFacet.sol, Governance.sol`

Description: Except for the respective initialization functions, all functions in both `BarnFacet` and `Governance` should require proper initialization, i.e., `require(ds.initialized)` in `BarnFacet`, and `require(isInitialized)` in `Governance`). Currently, that is not the case. Hence, users who call functions by means of the proxies will essentially waste gas; additionally, unforeseen side effects could occur.

Recommendation: For each function in `BarnFacet.sol`, except for `initBarn`, add the following pre-condition:

```
LibBarnStorage.Storage storage ds = LibBarnStorage.barnStorage();
require(ds.initialized, "BarnFacet has not been initialized");
```

For each function in `Governance.sol`, except for `initialize`, add the following pre-condition:

```
require(isInitialized, "Governance has not been initialized");
```

QSP-6 Unlocked Pragma

Severity: Informational

Status: Unresolved

File(s) affected: `Many Contracts`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-7 Unchecked function arguments

Severity: Informational

Status: Unresolved

File(s) affected: `Parameters.sol, Governance.sol, Rewards.sol, BarnFacet.sol, Barn.sol`

Description: The following functions do not ensure that certain arguments have non-zero values, which may lead to either incorrect contract initialization, or accidental misuse of functions.

1. In `Parameters.setMinQuorum`, `quorum` can be set to zero.
2. In `Governance.initialize`, `barnAddr` should be checked to be non-zero.
3. In `Rewards.constructor`, all address arguments should be checked to be non-zero.
4. `Rewards.setupPullToken` does not validate that address arguments are non-zero and the amount is non-zero.
5. `BarnFacet.initBarn` does not check that address `_bond` is non-zero.
6. Barn.constructor`should check if`address _owner` is non-zero.

Recommendation: Ensure all mentioned function arguments are sanitized accordingly.

QSP-8 Unchecked return value from `executeTransaction`

Severity: Informational

Status: Unresolved

File(s) affected: `Governance.sol`

Description: In the function `execute`, the `returnData` (nor the `success` boolean) associated with each call to `executeTransaction`. It is not clear if this is intentional.

Recommendation: Clarify if the return values or `success` status of each `executeTransaction` call should be checked.

QSP-9 `setupPullToken` should not be invoked multiple times

Severity: Informational

Status: Unresolved

File(s) affected: `Rewards.sol`

Description: If `setupPullToken` is invoked multiple times with different non-zero `source` arguments, existing multipliers will still exist. This may cause erroneous calculations when distributing rewards or updating any of the user-related state variables.

Recommendation: Disable multiple calls to the function, possibly only allowing subsequent calls to set `source = address(0)` to disable the feature.

QSP-10 Users without voting power are still flagged as having voted

Severity: Informational

Status: Unresolved

File(s) affected: `Governance.sol`

Description: If a user has no voting power, he will not add any votes to a proposal; nonetheless, the `castVote` function will still flag such that such a user has voted, which seems inconsistent.

Recommendation: Require that users to cast a vote, they must have a voting power greater than zero.

QSP-11 Privileged Roles and Ownership

Severity: Informational

Status: Unresolved

File(s) affected: `DiamondCutFacet.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
The function `diamondCut` in `DiamondCutFacet.sol` allows owner of the contract to delegate call any external arbitrary code, which includes but is not limited to: 1) transferring all the assets stored in the `Barn` contract; 2) draining every user wallets and contracts that set unlimited allowance of any ERC20 contract to this `Barn` contract.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

QSP-12 Unclear use-case for `receive` function in `Barn.sol`

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `Barn.sol`

Description: Given that most functionality is handled through the `fallback` function, it is not clear why the `receive` function is needed.

Recommendation: Clarify the intended use-case of the `receive` function.

QSP-13 Unclear timestamp setup logic in `propose`

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `Governance.sol`

Description: The `propose()` function uses `block.timestamp - 1` as the reference timestamp, instead of `block.timestamp`. It is unclear why this is, as the code does not provide proper documentation. Hence, this could have undetermined side effects for which we cannot assess at this point.

Recommendation: We recommend verifying whether the timestamp is properly set; if it is, we advise clarifying developers' rationale by adding documentation to the code.

Automated Analyses

Slither

Slither does not appear to handle the `ds.slot` assembly code used in the Diamond logic, and therefore could not be run at this time.

Mythril

Mythril warns of several uses of `block.timestamp` when computing past balance and stake values (e.g., in `stakeAtTs`), however we classified these as false positives.

Adherence to Specification

The code generally adheres to the provided specifications. However, we recommend further developing the inline documentation and specification.

Code Documentation

1. In `BarnFacet.sol` (L294), comment mentions "userDidDelegate". No such function exists. It should be "userDelegatedTo".
2. The diagram in `SPEC.md` mentions that a proposal has the following seven states: `Warm-up`, `Voting`, `Accepted`, `Failed`, `Queued for execution`, `Executed`, and `Expired`. However, the corresponding `struct` in `Governance.sol` lists nine states, including two that lack any documentation in the spec `Active` and `Grace`. It seems `Active` maps to `Voting`. If that is the case, make names consistent across the board.
3. In `LibDiamond.sol: L141`: "LibDiamondCut: _init is address(0) but_calldata is not empty" -- should add a space right after "but".

Adherence to Best Practices

1. In `Rewards.sol`, if the `source` address is set to `address(0)`, the rewards functionality is intended to be disabled. However, every call to `registerUserAction` will still invoke the logic in `_calculateOwed` (including the call to `ackFunds`). It is not clear if/why this is necessary. It would be more efficient to return immediately in `registerUserAction` if `source == address(0)`.
2. In the function `setContractOwner` in `LibOwnership.sol`, make sure that `_newOwner` is different from the existing one; if so, revert.
3. In `BarnFacet.sol`, the binary search logic is cloned in three locations; we suggest factoring out that logic into a reusable internal function.
4. In `BarnFacet.sol` (L350), naming the `Stake[]` array as `checkpoints` seem incorrect. Consider renaming that to `userStakingHistory`.
5. Some functions in `BarnFacet.sol` are redundant - for instance, `lock` and `_lock`. Consider removing all redundant functions.
6. In `Parameters.sol`, use double quotes for string literals.
7. In `Governance.sol`, the `propose()` function should make sure that the title and description parameters are not empty strings.
8. In `Parameters.sol: L38, L39, L45`, the conditionals do not accept "equal to" conditions; either the revert message or the code should be modified.

Test Results

Test Suite Results

We recommend adding instructions to the `README.md` on how to run tests and coverage.

```
Governance
  General tests
     ✓ should be deployed
  activate
     ✓ reverts if threshold not yet met (42ms)
     ✓ activates if threshold is met (63ms)
     ✓ reverts if already activated (72ms)
  propose
     ✓ create new proposal revert reasons (369ms)
     ✓ create new proposal (281ms)
     ✓ start vote && quorum (155ms)
     ✓ cast, cancel and change vote (416ms)
     ✓ cannot vote when vote is closed (255ms)
     ✓ verify proposal state (497ms)
     ✓ cannot execute proposals that are not queued (118ms)
     ✓ test proposal execution in queued mode (331ms)
     ✓ cannot cancel expired, failed or executed proposals (450ms)
     ✓ fail for invalid quorum (191ms)
     ✓ fail for invalid minimum threshold (352ms)
     ✓ test change periods (444ms)
     ✓ proposer cancel proposal (103ms)
     ✓ allows anyone to cancel a proposal if creator balance fell below threshold (166ms)
  cancellation proposal
     ✓ reverts if proposal id is not valid
     ✓ works only if proposal is in queued state (580ms)
     ✓ fails if user does not voting power above threshold (260ms)
    voting
       ✓ reverts for invalid proposal id
       ✓ reverts if cancellation proposal is not created (92ms)
       ✓ reverts if cancellation proposal expired (288ms)
       ✓ reverts if user tries to double vote (275ms)
       ✓ updates the amount of votes (311ms)
       ✓ allows user to change vote (332ms)
       ✓ changes initial proposal state to cancelled if accepted (385ms)
       ✓ does not change initial proposal state if not accepted (353ms)
    cancel vote
       ✓ reverts if cancellation proposal is not created (96ms)
       ✓ reverts if cancellation proposal expired (246ms)
       ✓ reverts if user tries to cancel vote if not voted (261ms)
       ✓ allows users to cancel their votes (315ms)
    executeCancellationProposal
       ✓ reverts if proposal state is not canceled (88ms)
       ✓ reverts if cancellation proposal failed (334ms)
       ✓ works if cancellation proposal was accepted (371ms)


36 passing (10s)

Barn
  General tests
     ✓ should be deployed
  deposit
     ✓ reverts if called with 0 (42ms)
     ✓ reverts if user did not approve token (41ms)
     ✓ calls registerUserAction on rewards contract (125ms)
     ✓ stores the user balance in storage (102ms)
     ✓ transfers the user balance to itself (96ms)
     ✓ updates the total of bond locked (93ms)
     ✓ updates the delegated user's voting power if user delegated his balance (213ms)
     ✓ works with multiple deposit in same block (208ms)
  depositAndLock
     ✓ calls deposit and then lock (118ms)
  balanceAtTs
     ✓ returns 0 if no checkpoint
     ✓ returns 0 if timestamp older than first checkpoint (81ms)
     ✓ return correct balance if timestamp newer than latest checkpoint (80ms)
     ✓ returns correct balance if timestamp between checkpoints (202ms)
  bondStakedAtTs
     ✓ returns 0 if no checkpoint
     ✓ returns 0 if timestamp older than first checkpoint (80ms)
     ✓ returns correct balance if timestamp newer than latest checkpoint (80ms)
     ✓ returns correct balance if timestamp between checkpoints (203ms)
  withdraw
     ✓ reverts if called with 0
     ✓ reverts if user does not have enough balance
     ✓ calls registerUserAction on rewards contract (70ms)
     ✓ sets user balance to 0 (138ms)
     ✓ does not affect old checkpoints (129ms)
     ✓ transfers balance to the user (150ms)
     ✓ updates the total of bond locked (137ms)
     ✓ updates the delegated user's voting power if user delegated his balance (182ms)
  lock
     ✓ reverts if timestamp is more than MAX_LOCK (108ms)
     ✓ reverts if user does not have balance
     ✓ reverts if user already has a lock and timestamp is lower (112ms)
     ✓ sets lock correctly (105ms)
     ✓ allows user to increase lock (120ms)
     ✓ does not block deposits for user (147ms)
     ✓ blocks withdrawals for user during lock (176ms)
  multiplierAtTs
     ✓ returns expected multiplier (101ms)
  votingPower
     ✓ returns raw balance if user did not lock (82ms)
     ✓ returns adjusted balance if user locked bond (101ms)
  votingPowerAtTs
     ✓ returns correct balance with no lock (181ms)
     ✓ returns correct balance with lock (210ms)
     ✓ returns voting power with decaying bonus (281ms)
  delegate
     ✓ reverts if user delegates to self
     ✓ reverts if user does not have balance (112ms)
     ✓ sets the correct voting powers for delegate and delegatee (124ms)
     ✓ sets the correct voting power if delegatee has own balance (188ms)
     ✓ sets the correct voting power if delegatee receives from multiple users (284ms)
     ✓ records history of delegated power (375ms)
     ✓ does not modify user balance (112ms)
     ✓ works with multiple calls in the same block (200ms)
  stopDelegate
     ✓ removes delegated voting power from delegatee and returns it to user (198ms)
     ✓ preserves delegate history (190ms)
     ✓ does not change any other delegated balances for the delegatee (269ms)
  events
     ✓ emits Deposit on call to deposit() (64ms)
     ✓ emits Deposit & DelegatedPowerIncreased on call to deposit() with delegated power (154ms)
     ✓ emits Withdraw on call to withdraw() (120ms)
     ✓ emits Withdraw & DelegatedPowerDecreased on call to withdraw() with delegated power (153ms)
     ✓ emits correct events on delegate (171ms)
     ✓ emits Lock event on call to lock() (85ms)
  multiplierOf
     ✓ returns the current multiplier of the user (100ms)

Diamond
  General tests
     ✓ should be deployed
  DiamondLoupe
     ✓ has correct facets (41ms)
     ✓ has correct function selectors linked to facet (96ms)
     ✓ associates selectors correctly to facets (111ms)
     ✓ returns correct response when facets() is called (62ms)
  DiamondCut
     ✓ fails if not called by contract owner
     ✓ allows adding new functions (301ms)
     ✓ allows replacing functions (162ms)
     ✓ allows removing functions (179ms)
  ownership
     ✓ returns owner
     ✓ reverts if transferOwnership not called by owner
     ✓ allows transferOwnership if called by owner

Rewards
  General
     ✓ should be deployed
     ✓ sets correct owner
     ✓ can set pullTokenFrom if called by owner
     ✓ can set barn address if called by owner
  ackFunds
     ✓ calculates the new multiplier when funds are added (99ms)
```

```
      ✓ does not change multiplier on funds balance decrease but changes balance (119ms)
    registerUserAction
      ✓ can only be called by barn (47ms)
      ✓ does not pull bond if function is disabled (134ms)
      ✓ does not pull bond if already pulled everything (97ms)
      ✓ updates the amount owed to user but does not send funds (47ms)
    claim
      ✓ reverts if user has nothing to claim
      ✓ transfers the amount to user (130ms)
      ✓ works with multiple users (209ms)
      ✓ works fine after claim (325ms)


  83 passing (13s)
```

Code Coverage

The code is generally well-covered by the test suite.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	94.27	83.9	91.67	94.3	
Bridge.sol	94.12	62.5	100	94.12	33
Governance.sol	93.94	84.31	88	93.94	… 416,425,426
Parameters.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IBarn.sol	100	100	100	100	
IBridge.sol	100	100	100	100	
All files	94.27	83.9	91.67	94.3	
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Barn.sol	100	100	100	100	
Rewards.sol	100	100	100	100	
contracts/facets/	99.09	96.25	97.14	99.13	
BarnFacet.sol	100	97.06	100	100	
DiamondCutFacet.sol	100	100	100	100	
DiamondLoupeFacet.sol	96	91.67	80	96.77	159,160
OwnershipFacet.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IBarn.sol	100	100	100	100	
IDiamondCut.sol	100	100	100	100	
IDiamondLoupe.sol	100	100	100	100	
IERC165.sol	100	100	100	100	
IERC173.sol	100	100	100	100	
IRewards.sol	100	100	100	100	
contracts/libraries/	85.51	45.65	92.31	85.53	
LibBarnStorage.sol	100	100	100	100	
LibDiamond.sol	85	45.24	100	85.94	… 152,154,156
LibDiamondStorage.sol	100	100	100	100	
LibOwnership.sol	85.71	50	75	75	28,29
All files	**96.56**	**80.82**	**96.61**	**96.48**	

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

```
b29ec12e219c1443a9d78388a4f14a36c4beb1fca9a05ed8ce6081a2e385e6c6 ./Bridge.sol

8324791c58b3fd8333c8917592e89c844a5a324b984a9b200299bcd475d0d547 ./Governance.sol

6a99e4d9bd073b477d1d20b2aac87c0ab429f22bd0b5b19898206e0d0fcd41a7 ./Parameters.sol

12fb8ecd6e0cb441afa121f1dd3b259050f0861a935224af28e56b970f9d2742 ./IBridge.sol

5050648db1b91725ba2fef46c162157ea527eab43704c8e90982fa7899c1f1bc ./IBarn.sol

9fa14dfbb3998693faacfbe7076ea4c566f3cb035dbdac61b6c4f2b2d65ee05b ./BarnMock.sol

73b9ba75e7fdb9111a9777eea08208de33beeb9e64651aa613dc81f520a29a19 ./ERC20Mock.sol

50babc4ad689ee42741b62b6395e9e5a3c5707beceea95b922e2242e64d2efe7 ./contracts/Rewards.sol

87079a356092c07198d1c6b252d58009cc2640a2b14cef04eb42e73bb9f63a61 ./contracts/Barn.sol

a3fc7ec422fc3c3aacef844150de1011527f967f963d817f5900b166c2498369 ./contracts/interfaces/IRewards.sol

b10478d3100451b3b15b647cadab5a1d4e201d661bf765a733c934db525fb2dc ./contracts/interfaces/IDiamondCut.sol

b3d97afef358c8b7d27af1def3fe62e022bbbf0cb0775682a83f24fba4b7767d ./contracts/interfaces/IBarn.sol

8fc8d2236a887f789cbfde1adc4ff60c4b21db07de2fa3d61fc1d8f281b2b27a ./contracts/interfaces/IDiamondLoupe.sol

6060daa89afce37b0030b402e5caaef1633a66d7c0d9e073185ec38f7ed26d01 ./contracts/interfaces/IERC165.sol

27de63177b363670fa8cb34123e47cb1f79f3ba2ef16c897d90a44d9148fbe5f ./contracts/interfaces/IERC173.sol

3856dc6e12944e2138245cd1bdf99391e8664b027c34cfc00573c8d60635085c ./contracts/libraries/LibBarnStorage.sol

2b2a7510f69972266775a32d6594652ae7505cab6931a0b58b20dafcc998ecd7 ./contracts/libraries/LibDiamond.sol

dc8e7deae762999c542df327f8fd84e30fee99c316409e8b01e5068e50f309ad ./contracts/libraries/LibOwnership.sol

ddd3c4f5b0c13806b1151e93293f1518160f9760397dfcdcdd7147e777495bf6 ./contracts/libraries/LibDiamondStorage.sol

1768f83d812bc44810faa1d91ccbb4d3327502bb77f17e770fb1cd02349abec9 ./contracts/facets/DiamondCutFacet.sol

aa38430ca1893b5ad7669b7f826cbe9597110a65acb0733779a556af6ef18dcf ./contracts/facets/OwnershipFacet.sol

96e08f189372273c31de74b35a2e28329b8f11d2f2b0b01391e378c7dd76babc ./contracts/facets/BarnFacet.sol

068f240c3d03ab33d5391f9e38258c8af871e4b3d180d9ebe3f6334eacb0e75c ./contracts/facets/DiamondLoupeFacet.sol

710fe6dc655dc941148b2abc1fc4899f808e574668bc9dfc7356656c3e1242b6 ./contracts/mocks/Test2Facet.sol

9f4736bef5f0ae14b3e79972ea071f4651c61a9bd2932cf778ca91ddcc47ff15 ./contracts/mocks/BarnMock.sol

945076b0771a774cfb3ae4327b894cf08315233a771d8cfdc9cea4607abf22e0 ./contracts/mocks/ERC20Mock.sol

7db465ddd5df441fba58879880a0b4cd5c1e288c0d0129fab263f05fcff55bf5 ./contracts/mocks/Test1Facet.sol

f45c195d2808cf3b14913c093595844a593fe3e4d5ddfb6dcd2da02bb7976558 ./contracts/mocks/RewardsMock.sol

3e6ea95285aa84a2a65b3aa8667c86951c875f443bfaf4bc9c4a67e4e069a591 ./contracts/mocks/MulticallMock.sol
```

Tests

```
c17c182d4ee6e030e8803ce3c462e7c8eb62bbf664c95a924eb6ea66ff712299 ./Barn.test.ts

47c4cb3741985dc09a1c7d52db908302e434511026fcfd0cee3c870ab2db6ebd ./diamond.test.ts

05a228611e0c24d1e533f2056f79a12cb84f8628524cc8611adee25f073a50d5 ./Rewards.test.ts

d281bcfb0e604f21186c73933841a6476a4e890687a3e8a4452b85d9268eea04 ./helpers.ts

c782ffec1253aa2bac09551df2cd711fe028ffbc230b9afd3b9381e000288134 ./diamond.ts

c567be775fbaa37b38f42e601792f53f0e7e58360d2ded874125b1173b1aafa3 ./deploy.ts

83bb7c36e31071a9d3638e7c72684129da9f1ba94260f55dd8c43aee57597a38 ./time.ts

81e4ec029b7d9a8af29a0567dfb3f9218b3f087de8be912a887166d277f67043 ./test/helpers.ts

7e3192d257c45cf6266ba0f006a9dd01e486e30c59d5e07b73a20e1a4e163f3f ./test/Governance.test.ts
```

Changelog

- 2021-01-21 - Initial report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.