Quantstamp Security Assessment Certificate

February 10th 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 2021-02-09
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** (4 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**5** (1 Resolved)
Informational Risk Issues	**6** (3 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.
Update: All issues have been either resolved or acknowledged by the Barn Bridge team based on recent commits for DAO (`47b14a6`) and Barn (`0166325`).

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

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: Acknowledged

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.
Update: This has been acknowledged by the Barn Bridge team, and can only happen if `P_1` and `P_2` are in the same block.

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

Severity: *Low Risk*

Status: Fixed

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: Acknowledged

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)`.
Update: This has been confirmed as intended behavior.

QSP-4 Unchecked `transferFrom` return value

Severity: *Low Risk*

Status: Acknowledged

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.
Update from the Barn Bridge team: Barn only interacts with our BOND token which is implemented using the OZ ERC20 which always return true for `transfer` and `transferFrom`.

QSP-5 Functions do not check if contract is initialized

Severity: *Low Risk*

Status: Acknowledged

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”);
```

Update from the Barn Bridge team: We will initialize the contracts immediately after deploy so it should not be a problem.

QSP-6 Privileged roles and ownership

Severity: *Informational*

Status: Acknowledged

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.
Update from the Barn Bridge team: The DAO will be the owner of the BARN contract. Any changes will have to go through the DAO.

QSP-7 Unlocked Pragma

Severity: *Informational*

Status: Fixed

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-8 Unchecked function arguments

Severity: *Informational*

Status: Fixed

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-9 Unchecked return value from `executeTransaction`

Severity: *Informational*

Status: Acknowledged

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.
Update from the Barn Bridge team: The success is checked inside `executeTransaction` and reverts if value returned is false.

QSP-10 `setupPullToken` should not be invoked multiple times

Severity: *Informational*

Status: Acknowledged

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.
Update from the Barn Bridge team: This is intended behavior (e.g. we may want to do some changes to the rewards amount mid flight). User multipliers are not a problem because the contract should distribute any BOND tokens that are reaching its balance so they should never be reset.

QSP-11 Users without voting power are still flagged as having voted

Severity: *Informational*

Status: Fixed

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

Severity: *Undetermined*

Status: Acknowledged

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.
Update from the Barn Bridge team: The fallback function requires a `msg.sig` that is defined on one of the facets; it reverts if not found. The `receive` function is used to avoid running the fallback function on simple ETH transfers.

QSP-13 Unclear timestamp setup logic in `propose`

Severity: *Undetermined*

Status: Acknowledged

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.
Update from the Barn Bridge team: `block.timestamp - 1` is used to avoid any flash loan attacks.

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". **Update:** fixed.
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 `Governace.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". **Update:** fixed.

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)`. **Update:** fixed.
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. **Update:** fixed.
4. In `BarnFacet.sol` (L350), naming the `Stake[]` array as `checkpoints` seem incorrect. Consider renaming that to `userStakingHistory`. **Update:** fixed.
5. Some functions in `BarnFacet.sol` are redundant - for instance, `lock` and `_lock`. Consider removing all redundant functions. **Update:** fixed.

6. In `Parameters.sol`, use double quotes for string literals. **Update:** fixed.
7. In `Governance.sol`, the `propose()` function should make sure that the title and description parameters are not empty strings. **Update:** fixed.
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. **Update:** fixed except for `threshold > 50, "Minimum is 50."`.

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 (47ms)
     ✓ activates if threshold is met (63ms)
     ✓ reverts if already activated (56ms)
  propose
     ✓ create new proposal revert reasons (397ms)
     ✓ create new proposal (283ms)
     ✓ start vote && quorum (169ms)
     ✓ cast, cancel and change vote (436ms)
     ✓ castVote fails if user does not have voting power (132ms)
     ✓ cannot vote when vote is closed (226ms)
     ✓ verify proposal state (518ms)
     ✓ cannot execute proposals that are not queued (127ms)
     ✓ test proposal execution in queued mode (332ms)
     ✓ cannot cancel expired, failed or executed proposals (490ms)
     ✓ fail for invalid quorum (197ms)
     ✓ fail for invalid minimum threshold (380ms)
     ✓ test change periods (468ms)
     ✓ proposer cancel proposal (113ms)
     ✓ allows anyone to cancel a proposal if creator balance fell below threshold (148ms)
     ✓ allows cancellation only when proposal is in warmup or active state (374ms)
  abrogation proposal
     ✓ reverts if proposal id is not valid
     ✓ works only if proposal is in queued state (605ms)
     ✓ fails if user does not voting power above threshold (268ms)
  voting
     ✓ reverts for invalid proposal id
     ✓ reverts if abrogation proposal is not created (97ms)
     ✓ reverts if abrogation proposal expired (287ms)
     ✓ reverts if user does not have voting power (272ms)
     ✓ reverts if user tries to double vote (280ms)
     ✓ updates the amount of votes (315ms)
     ✓ allows user to change vote (339ms)
     ✓ changes initial proposal state to cancelled if accepted (438ms)
     ✓ does not change initial proposal state if not accepted (362ms)
  cancel vote
     ✓ reverts if abrogation proposal is not created (96ms)
     ✓ reverts if abrogation proposal expired (268ms)
     ✓ reverts if user tries to cancel vote if not voted (258ms)
     ✓ allows users to cancel their votes (329ms)
  abrogateProposal
     ✓ reverts if proposal state is not canceled (91ms)
     ✓ reverts if abrogate proposal failed (351ms)
     ✓ works if abrogation proposal was accepted (421ms)
  stored parameters
     ✓ stores parameters on proposal on creation (124ms)
     ✓ parameters changed mid-flight do not affect running proposals (810ms)


  41 passing (12s)

  Barn
    General tests
       ✓ should be deployed
    deposit
       ✓ reverts if called with 0
       ✓ reverts if user did not approve token
       ✓ calls registerUserAction on rewards contract (131ms)
       ✓ stores the user balance in storage (117ms)
       ✓ transfers the user balance to itself (85ms)
       ✓ updates the total of bond locked (68ms)
       ✓ updates the delegated user's voting power if user delegated his balance (148ms)
       ✓ works with multiple deposit in same block (150ms)
       ✓ does not fail if rewards contract is set to address(0) (80ms)
    depositAndLock
       ✓ calls deposit and then lock (90ms)
    balanceAtTs
       ✓ returns 0 if no checkpoint
       ✓ returns 0 if timestamp older than first checkpoint (65ms)
       ✓ return correct balance if timestamp newer than latest checkpoint (67ms)
       ✓ returns correct balance if timestamp between checkpoints (158ms)
    bondStakedAtTs
       ✓ returns 0 if no checkpoint
       ✓ returns 0 if timestamp older than first checkpoint (64ms)
       ✓ returns correct balance if timestamp newer than latest checkpoint (61ms)
       ✓ returns correct balance if timestamp between checkpoints (173ms)
    withdraw
       ✓ reverts if called with 0
       ✓ reverts if user does not have enough balance
       ✓ calls registerUserAction on rewards contract (146ms)
       ✓ sets user balance to 0 (147ms)
       ✓ does not affect old checkpoints (105ms)
       ✓ transfers balance to the user (114ms)
       ✓ updates the total of bond locked (105ms)
       ✓ updates the delegated user's voting power if user delegated his balance (129ms)
    lock
       ✓ reverts if timestamp is more than MAX_LOCK (78ms)
       ✓ reverts if user does not have balance
       ✓ reverts if user already has a lock and timestamp is lower (80ms)
       ✓ sets lock correctly (71ms)
       ✓ allows user to increase lock (94ms)
       ✓ does not block deposits for user (100ms)
       ✓ blocks withdrawals for user during lock (131ms)
    multiplierAtTs
       ✓ returns expected multiplier (80ms)
    votingPower
       ✓ returns raw balance if user did not lock (67ms)
       ✓ returns adjusted balance if user locked bond (74ms)
    votingPowerAtTs
       ✓ returns correct balance with no lock (131ms)
       ✓ returns correct balance with lock (148ms)
       ✓ returns voting power with decaying bonus (185ms)
    delegate
       ✓ reverts if user delegates to self
       ✓ reverts if user does not have balance (85ms)
       ✓ sets the correct voting powers for delegate and delegatee (93ms)
       ✓ sets the correct voting power if delegatee has own balance (146ms)
       ✓ sets the correct voting power if delegatee receives from multiple users (212ms)
       ✓ records history of delegated power (283ms)
       ✓ does not modify user balance (80ms)
       ✓ works with multiple calls in the same block (144ms)
    stopDelegate
       ✓ removes delegated voting power from delegatee and returns it to user (162ms)
       ✓ preserves delegate history (123ms)
       ✓ does not change any other delegated balances for the delegatee (187ms)
    events
       ✓ emits Deposit on call to deposit() (55ms)
       ✓ emits Deposit & DelegatedPowerIncreased on call to deposit() with delegated power (118ms)
       ✓ emits Withdraw on call to withdraw() (84ms)
       ✓ emits Withdraw & DelegatedPowerDecreased on call to withdraw() with delegated power (113ms)
       ✓ emits correct events on delegate (132ms)
```

```
      ✓ emits Lock event on call to lock() (70ms)
    multiplierOf
      ✓ returns the current multiplier of the user (89ms)

  Diamond
    General tests
      ✓ should be deployed
    DiamondLoupe
      ✓ has correct facets
      ✓ has correct function selectors linked to facet (42ms)
      ✓ associates selectors correctly to facets (60ms)
      ✓ returns correct response when facets() is called (38ms)
    DiamondCut
      ✓ fails if not called by contract owner
      ✓ allows adding new functions (172ms)
      ✓ allows replacing functions (125ms)
      ✓ allows removing functions (104ms)
    ownership
      ✓ returns owner
      ✓ reverts if transferOwnership not called by owner
      ✓ reverts if transferOwnership called with same address
      ✓ allows transferOwnership if called by owner

  Rewards
    General
      ✓ should be deployed
      ✓ sets correct owner
      ✓ can set pullTokenFrom if called by owner
      ✓ sanitizes the parameters on call to setPullToken
      ✓ can set barn address if called by owner
      ✓ reverts if setBarn called with 0x0
    ackFunds
      ✓ calculates the new multiplier when funds are added (93ms)
      ✓ does not change multiplier on funds balance decrease but changes balance (119ms)
    registerUserAction
      ✓ can only be called by barn (46ms)
      ✓ does not pull bond if function is disabled (125ms)
      ✓ does not pull bond if already pulled everything (83ms)
      ✓ updates the amount owed to user but does not send funds (52ms)
    claim
      ✓ reverts if user has nothing to claim
      ✓ transfers the amount to user (122ms)
      ✓ works with multiple users (186ms)
      ✓ works fine after claim (305ms)


  87 passing (11s)
```

Code Coverage

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

Update: As of the latest changes, we could not run coverage for the Barn repository due to the following issue (`npm run test` was unaffected):

```
Istanbul reports written to ./coverage/ and ./coverage.json
Error in plugin solidity-coverage: TSError:  ✓ Unable to compile TypeScript:
test/Barn.test.ts:5:60 - error TS2305: Module '"../typechain"' has no exported member 'ChangeRewardsFacet'.

5 import { BarnFacet, Erc20Mock, RewardsMock, MulticallMock, ChangeRewardsFacet } from '../typechain';
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	94.84	83.33	92.11	94.86	
Bridge.sol	94.12	62.5	100	94.12	33
Governance.sol	94.54	85.34	88.89	94.54	… 461,470,471
Parameters.sol	100	78.57	100	100	
contracts/interfaces/	100	100	100	100	
IBarn.sol	100	100	100	100	
IBridge.sol	100	100	100	100	
All files	**94.84**	**83.33**	**92.11**	**94.86**	

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

```
46cb7a7e3f6b1bf863973a858515ee44ccb29a1a67e890166b3c501fae1a61d3 ./Bridge.sol
9c6b4644461a0353e6642c0e740797fec4156ef8cc50cfafc1576d9780eb7d16 ./Governance.sol
93822dc5b49672f8b8d14540783a3aaee6caca603a0cab13b1867a147a016180 ./Parameters.sol
09e5c2b5dec1120a4b6f967f2f463d72def2c798fa80a94344d7e6c75166440b ./IBridge.sol
90c942c485f3b6615f5ef4c1c67ede8de420052c34ab1df8e128165bd5a93dad ./IBarn.sol
9fa14dfbb3998693faacfbe7076ea4c566f3cb035dbdac61b6c4f2b2d65ee05b ./BarnMock.sol
73b9ba75e7fdb9111a9777eea08208de33beeb9e64651aa613dc81f520a29a19 ./ERC20Mock.sol
bfdd100c998788538a41d78ac36823407482b4af89c434965ff8576f3a292493 ./contracts/Rewards.sol
9d0e28ed66d85563615906475 45f5b9d8057cf9317ba5d74fc9e50cc6a0f96ef ./contracts/Barn.sol
46b3c78b244f0b029a58ea1a75b243cc133a6856e0ad4723e7f06d4cb5c8bf4f ./contracts/interfaces/IRewards.sol
b7c545ccba910ef35ded1168f0fc8ab47146f293c48d3b1ff3de979230e2e748 ./contracts/interfaces/IDiamondCut.sol
aefd498e68f56f77297eddfed4e40661b66d3946803a7f945ddd99805a30d5d3 ./contracts/interfaces/IBarn.sol
fc52ed8e2e567731cfb563110f28baa40ae87ebe60fdd6979e1691b668caed4f ./contracts/interfaces/IDiamondLoupe.sol
55ea35d5bf029e0997615e4f6c80a72706a5649c0cbb34b6e57a717703d53f8a ./contracts/interfaces/IERC165.sol
81f76fa7eb96ecf24afef5b3e0226fd15c9d09ecb8b392358b46a10bd2326b25 ./contracts/interfaces/IERC173.sol
7ef5228e237fc1c2bcbca0e7fb52d127c1b5f5646224ed591f20b41b6fe105c1 ./contracts/libraries/LibBarnStorage.sol
2d3ec0d75f7e94659bba1354d4a0e8c69b5aa5761bd61ead3a993ca25752bdd5 ./contracts/libraries/LibDiamond.sol
6398965a2c62222271528262ea4f0633e6ce6727f1e9a442825188f6d819b0af ./contracts/libraries/LibOwnership.sol
8a2b91b55b3785f57e58e1fbda70372e776443cc792b22e54c3a40fb765a90bb ./contracts/libraries/LibDiamondStorage.sol
4741a61b374f1cf16cc91227337ff71c4cfa7b217340c6ac35cf399523530396 ./contracts/facets/DiamondCutFacet.sol
9c9438be8b593907a098bb71dfe0a97afca9db7be4d88c2bfb47d42e351cfb78 ./contracts/facets/OwnershipFacet.sol
7145e4ce8784d1a228c6469b954d732e26ef200f37ab905175cf618a7606486c ./contracts/facets/BarnFacet.sol
2018bc050dbd85de99d24b8325fe3171fd2ed55cc0d8c492991343d04d3c7f05 ./contracts/facets/ChangeRewardsFacet.sol
208ae289d5441e8e8cdf32abb550e43f784f8778572287eb90f1a57e77ce33c1 ./contracts/facets/DiamondLoupeFacet.sol
ad1c831bcf3d99e7c2870e4bc3282f310241ee73887c3fb1643c9723c54525ac ./contracts/mocks/Test2Facet.sol
080ce8e810e2d1ba5b96ef4e79cad66083ad88c2e03dde11c226335936ecd731 ./contracts/mocks/BarnMock.sol
281aa98137521944cbf3366eafc1893826d902795ec356ce830cb5b866e370d8 ./contracts/mocks/ERC20Mock.sol
ede800fc20d4d25eccc6a2edc3d9753b87484df214614da77f1677a4275d4659 ./contracts/mocks/Test1Facet.sol
95a13b08b1ab93010160eea941ca1b0ae4d5434a8e4fe8e5e3a4939dbeb4db79 ./contracts/mocks/RewardsMock.sol
ce2d8a18105bf6d2348ac62dd017ee8e5a2512a102d1e79717772eddb424a2cf ./contracts/mocks/MulticallMock.sol
```

Tests

```
81e4ec029b7d9a8af29a0567dfb3f9218b3f087de8be912a887166d277f67043 ./helpers.ts
5d863c3bb3c02bd6d250813711e7df6bf34bff87d147b1a4cadf8f3e3f543a2a ./Governance.test.ts
777960ebaf5aa06dc018a8aa3e3f3bbdec4c30d16fc5cc905368a4c808ea6368 ./test/Barn.test.ts
0d0cb5a6d67675532de7117cb78a496791f5c0a884544bd193a484ab014e91c6 ./test/diamond.test.ts
342ea9a9ae5f3d20d324f17ba64ef3f5f191ca5b3986a474fdceb8e6ac5be60e ./test/Rewards.test.ts
83532c3df1167b2328b1a175619ce609f5bd0784a69239fe0995fed790f01f0d ./test/helpers/helpers.ts
c782ffec1253aa2bac09551df2cd711fe028ffbc230b9afd3b9381e000288134 ./test/helpers/diamond.ts
c567be775fbaa37b38f42e601792f53f0e7e58360d2ded874125b1173b1aafa3 ./test/helpers/deploy.ts
83bb7c36e31071a9d3638e7c72684129da9f1ba94260f55dd8c43aee57597a38 ./test/helpers/time.ts
```

Changelog

- 2021-01-21 - Initial report
- 2021-02-09 - Updated report based on commit for DAO (`47b14a6`) and Barn (`0166325`)

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.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, 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, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. 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. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.