Quantstamp Security Assessment Certificate

July 22nd 2022 — Quantstamp Verified

Algem

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

Executive Summary

Type	Liquid Staking
Auditors	Marius Guggenmos, Senior Research Engineer Andy Lin, Senior Auditing Engineer Philippe Dumonet, Senior Research Engineer
Timeline	2022-06-24 through 2022-06-24
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Documentation
Documentation Quality	Medium
Test Quality	Low

Source Code

Repository	Commit
algem (initial audit)	ac254ea
algem (re-audit)	15f9daf
algem (re-audit 2)	b11a660

Total Issues	**36** (35 Resolved)
High Risk Issues	**8** (8 Resolved)
Medium Risk Issues	**7** (7 Resolved)
Low Risk Issues	**12** (12 Resolved)
Informational Risk Issues	**3** (3 Resolved)
Undetermined Risk Issues	**6** (5 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

Initial Audit:
The first audit iteration of the Algem code base revealed multiple severe defects that need to be addressed before deployment. The most severe ones are denial of service, locking funds in the contract, wrong rewards computation, and draining of contract funds. Additionally, the code base suffers from underspecification of functions and requires clean up of commented out code and debug functions.
The test coverage is critically low and the tests fail. The lack of proper testing shows in the number of high severity issues related to basic functionality of the contracts. We recommend focusing on this area specifically and in addition to general tests, implement tests that make sure the reported vulnerabilities are no longer present in the fixed code.
The Algem code base is not ready for deployment and should be completely re-audited once the findings in this report are addressed.

Re-audit Update: commit `15f9daf`
The team has refactored the code and tried to fix most of the issues. However, new problems appear due to the new logic. Also, we strongly recommend the team has thorough testing before launching.

Re-audit 2 Update: commit `b11a660`
The team has fixed or mitigated the newly reported issues. The code base has largely improved from the initial commit. Still, we recommend the team thoroughly test it before launching.

ID	Description	Severity	Status
QSP-1	Lack of Production Readiness	High	Mitigated
QSP-2	Contract Can Be Drained	High	Fixed
QSP-3	Current Supply Used for Calculations Rather than Historical Supply	High	Fixed
QSP-4	Staker Rewards Subject to Manipulation	High	Fixed
QSP-5	Wrong Proxy Address in Initialize	High	Fixed
QSP-6	Locked Funds Through Griefing	High	Fixed
QSP-7	Problems With Reward Computation	High	Mitigated
QSP-8	Unable to Mint `nSBY`	High	Fixed
QSP-9	Getting User Rewards Will Not Work if Era Missed	Medium	Fixed
QSP-10	Debug Functions in Production Contract	Medium	Fixed
QSP-11	Gas Concerns: User Rewards Might End Up Locked	Medium	Fixed
QSP-12	Assuming Fixed Exchange Rate on the LP Token and the DNT Token	Medium	Fixed
QSP-13	Risk of Denial of Service with Repeated DappStaking Calls	Medium	Fixed
QSP-14	Potentially Missing DApp Rewards	Medium	Fixed
QSP-15	`setup` Function Can Be Called More than Once	Low	Fixed
QSP-16	Duplicates Can Be Added to Arrays	Low	Fixed
QSP-17	Use of `address.transfer`	Low	Fixed
QSP-18	Adding Staker Uses Wrong Address for Balance	Low	Fixed
QSP-19	Gas Concerns: `updateAll`	Low	Fixed
QSP-20	Hardcoded Initialization of `lastStaked` and `lastUnstaked`	Low	Fixed
QSP-21	Risk of Missing Variable Initialization	Low	Fixed
QSP-22	Potential Balance Inconsistency	Low	Fixed
QSP-23	Gas Concerns: `nDistributor.sol`	Low	Mitigated
QSP-24	Risk of Issuing Wrong Utilities on Transfer	Low	Fixed
QSP-25	Lack of Events on Critical State Changes	Informational	Fixed
QSP-26	Unlocked Pragma	Informational	Mitigated
QSP-27	Unused or Unnecessary Variables and Functions	Informational	Fixed
QSP-28	Role Change without Manager/Owner Update	Undetermined	Acknowledged
QSP-29	Lack of Ability to Remove LP Tokens	Undetermined	Fixed
QSP-30	Unclear Revenue Accounting	Undetermined	Fixed
QSP-31	Lacking Access Control	Undetermined	Fixed
QSP-32	Limited Integrity Of Code Supply Chain	Undetermined	Fixed
QSP-33	Problems with New Reward Computation	Medium	Fixed
QSP-34	Risk of the `sync` Function	Low	Fixed
QSP-35	Potential Inconsistency on `lphandlers` and `lptokens`	Low	Fixed
QSP-36	Catching All Exceptions	Undetermined	Mitigated

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:
Contracts in the `packages/hardhat/libs` directory are not within the scope of the audit.

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 .`

Findings

QSP-1 Lack of Production Readiness

Severity: *High Risk*

Status: Mitigated

Description: As described in the summary, the project is lacking in many areas and currently cannot be considered as ready for production. The following is a non-exhaustive list of issues regarding the production-readiness:

- Commented out code, which leads to empty modifiers and functions
- Unused contract storage variable such as `rewardsByAddress` (`LiquidStaking.sol`)
- Lacking specification
- Functions documented as being for "debug purposes" (L129-L144, `LiquidStaking.sol`)
- Limited and/or non-descriptive documentation
- Limited tests that are broken

- `eraRevenue` unused outside of `fillPools` (only 1/10th seems to be used)

Recommendation: Before thinking about deploying these contracts, address all of the issues in this report and conduct a full re-audit.

Update: The team has cleaned up and refactored the code. However, Quantstamp strongly recommends that the team add more tests and run it on the test net for a period to confirm all features are working as expected before fully launching the main Astar network.

QSP-2 Contract Can Be Drained

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The `withdraw` function does not check whether a `Withdrawal` has already been claimed, as it only sets the `eraReq` to `0`. As a result, once the necessary number of eras pass, a malicious user could repeatedly call `withdraw` with the respective ID until the contract is drained. Similarly, the `claim` function can be called repeatedly by an address in order to drain the contract, as there is no logic to ensure that a user doesn't claim rewards that have already been claimed.

Recommendation: Ensure that claims are accounted for and prevent users from over-claiming.

Update: The team has fixed the `withdraw` function by adding a check that the specific withdraw cannot be called twice. For the `claim` function, it checks against the `totalUserRewards` so a user can at most claim the rewarded amount.

QSP-3 Current Supply Used for Calculations Rather than Historical Supply

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: In the function `getUserRewards` the DNT token's current total supply is used for the rewards calculation together with historical balances retrieved via `balanceOfAt`.

Exploit Scenario:

1. Attacker stakes large amount of tokens, minting DNT in the process, increasing balance and supply.
2. Manager takes a snapshot, balance and supply are stored.
3. Attacker unstakes a large amount, burning their DNT.
4. The `getUserRewards` method uses the smaller, current supply while looking at the staker's historical, larger balance, resulting in a larger overall value being output from the method.
5. The attacker calls the `claim` method claiming rewards far greater than what they were originally entitled to.

Recommendation: Fix the `getUserRewards` method to use the historical supply, this can be done similarly to `balanceOfAt`, by calling the `totalSupplyAt` provided by the `ERC20Snapshot` contract.

Update: The reward calculation has been re-designed, and the `getUserRewards` function now returns the stored `totalUserRewards` values instead.

QSP-4 Staker Rewards Subject to Manipulation

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The `globalClaim` method does not validate the `_era` parameter or the caller in any way. This allows for a malicious actor to set an era that's very far in the future or simply overwrite the `eraStakerReward` for an existing era. This can strongly diminish if not even destroy accrued value for the contract as the `fillPools` method relies on `eraStakerReward` to top-up the `rewardPool`.
Furthermore, combining multiple calls to the `globalClaim` method with calls to `fillPools` will cause the `unstakingPool` to be massively inflated. This is due to missing validation, checking whether the queried eras have been marked as done. Without such validation every call to `fillPools` with a certain era look-ahead would continuously increase the `unstakingPool` value by the respective `eraRevenue`.

Recommendation: Fix the logic and add tests to ensure that such bugs are caught.

Update: The `globalClaim` function and `fillPools` has become private functions. This reduce the chance being called aside from the `updateAll` and `sync` functions. Also, the `updateAll` function limits the `era` to be `currentEra() - 1` and calls the `globalClaim` and `fillPools` together. This reduces most of the risk. Aside from the above, the team adds a validation `_era < currentEra()` on the `globalClaim` function to further verify the `_era`.

QSP-5 Wrong Proxy Address in Initialize

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The `LiquidStaking.initialize` function sets the `proxyAddr` to `msg.sender`, which is not the address of the proxy but of the account that deployed the contract.

Recommendation: The correct address to use is `address(this)`. Since `address(this)` is available globally, we recommend removing the `proxyAddr` variable altogether.

Update: The team fixed the issue as recommended.

QSP-6 Locked Funds Through Griefing

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The functions `globalStake` and `globalUnstake` work under the assumption that they are called for eras that have already passed. Since anyone can call these functions with arbitrary arguments, an attacker can "close" an era prematurely, locking up any funds in the contract that are staked or unstaked after that call.

Exploit Scenario:

1. A new era just started.
2. An attacker stakes a small amount by calling `stake` and setting `eraStaked[currentEra()]` to a value greater than 0.
3. The attacker now calls `globalStake(currentEra())`. Since `eraStaked[currentEra()]` is non-zero, `sum2stake` will be non-zero. This means `lastStaked` will be set to `currentEra()`.
4. An unsuspecting user now calls stake and the amount is added to `eraStaked[currentEra()]`. However, calls to `globalStake` will never stake any funds from the current era, since the for loop starts at `currentEra() + 1`, locking the amount in the contract.

Recommendation: All of the update functions called in `updateAll` should be declared `private` or `internal` to avoid attackers messing with the current era.

Update: The team fixed the issue as recommended.

QSP-7 Problems With Reward Computation

Severity: *High Risk*

Status: Mitigated

File(s) affected: `contracts/LiquidStaking.sol`

Description: The function `getUserRewards` has several issues:

1. The current computation of the users share `(dntBalance + lpBalance) / totalDntBalance` will return more than 100 percent of the rewards.
2. The rewards can be boosted retroactively through the LP tokens. For example a user can use a flash loan to claim the rewards and receive significantly more that way, returning the LP tokens at the end of the transaction.
3. The DappsStaking API that is used for the total rewards is `read_era_reward`. From looking at the documentation for this function, it seems like this function returns the reward for the entire network and not just the current address.

Recommendation:

1. The extra rewards that can be claimed through LP tokens must be limited in some way to make sure some users don't end up without any rewards since the `rewardPool` is empty.
2. LP tokens usually do not support snapshots. Consequently, the team should provide an adapter contract in the middle between the user and the liquidity-providing platform, such as a DEX. The adapter contract can receive nSBY tokens and proxy the actions to the liquidity-providing media. In other words, the user's nSBY token can continue to earn staking rewards only if the user uses this adapter contract. Once locked in the adapter contract, the balance cannot be changed unless removing the liquidity. Whenever the balance changes in the adapter contract, it should call back to the `LiquidStaking` contract to update the rewards (see the recommendation of the "Gas Concerns: User Rewards Might End Up Locked" issue for how to update the rewards).
3. Verify whether the API really does what you believe it does. In case it returns the rewards for the whole network like the documentation suggests, consider computing the rewards of just the contract by calculating `read_era_reward(_era) * contractStake[_era] / read_era_staked(_era)`. `contractStake` will have to be filled using `read_staked_amount` in `globalStake`. Whether this works correctly will have to be verified as well since this approach was devised using just the documentation.

Update: The team changes the design to fix the issues regarding reward calculations. Note that new issues appear with the new implementation. However, to make issue-tracking easier and keep the description relevant, we will update the new problems as a new QSP issue and flag this as mitigated.
Following is the design explanation from the team:

```
To solve the problem, the counting logic has been changed. In the off-chain loop, the eraShot() function will be called for each user, generating data that will be supplemented in the next era at the first eraShot() and the
number of user awards will be written to the state. To do this, the following functions have been modified: eraShot(), getUserRewards(). Added function findMean().

To avoid front-running of nTokens, a buffer has been added to which all nTokens that the user receives fall into. Tokens are in the buffer until the next era and are not taken into account when calculating rewards.
It has not yet been possible to completely eliminate the possibility of an exploit when counting lp tokens. To mitigate the vulnerability, the off-chain loop will combine the count of user rewards into small bunches.

The solution to this problem greatly changed the logic and most of the QSPs became irrelevant.
```

QSP-8 Unable to Mint `nSBY`

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/nSBY.sol`, `contracts/nDistributor.sol`

Description: The `nSBY.sol:mintNote` function (L55-57) calls `ERC20.sol:_mint`. The `_mint` function from the OpenZeppelin contract will call `_beforeTokenTransfer(address(0), account, amount)`, and use `address(0)` as the `from` parameter of the `_beforeTokenTransfer` function. As a result, in `nDistributor.sol:_beforeTokenTransfer` (L86-93), it will call `distributor.transferDnt` with `from` as `address(0)`. The problem is in `nDistributor.sol:transferDnt` (L494-518), where it will check `users[_from].dnt[_dnt].dntInUtil[_utility] >= _amount`. For this check to pass, the `from` address needs to have the utility amount for the transfer to succeed. However, unless someone transferred the token to `address(0)`, the transaction will fail. Consequently, the `mintNote` function will fail under normal circumstances.

Recommendation: In `nSBY.sol:_beforeTokenTransfer`, skip calling `distributor.transferDnt` when `from` is equal to `address(0)`.

Update: The team fixed the issue as recommended.

QSP-9 Getting User Rewards Will Not Work if Era Missed

Severity: *Medium Risk*

Status: Fixed

Description: If the `eraShot` method is not called every era then the `getUserRewards` method will get a `snapshotId` of `0` when attempting to retrieve the era's snapshot ID. This will cause the entire method to revert as the `ERC20Snapshot` snapshot value retrieval methods explicitly revert if the snapshot ID is `0`. Depending on the frequency of eras the required uptime of the manager may pose a problem. Even if the manager presents the required uptime the critical blocks could be filled up with junk transactions by an attacker to prevent `eraShot` calling transactions from being included in time.

Recommendation: Adjust the logic such that not all eras have to be specifically snapshotted. This could be done by skipping an iteration if the `snapshotId` is `0` via `if (snapshotId == 0) continue;`.

Update: The team has re-design the calculation to not rely on a snapshot.

QSP-10 Debug Functions in Production Contract

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: Several functions in the `LiquidStaking` contract are documented with the string `debug purposes`. Debug functions should never be part of production level code.

Recommendation: Remove any debug function from the contract. If debug functions are required for testing, create a new contract in a `test` directory and inherit from the one you want to test.

Update: The team fixed the issue as recommended.

QSP-11 Gas Concerns: User Rewards Might End Up Locked

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The function `getUserRewards` might run out of gas if a user has not claimed any rewards for a long period of time since it performs external calls in a loop.

Recommendation: Consider extending `getUserRewards` by accepting an an end era instead of always looping from the time the user started staking up until the current era. This will require keeping track of which eras have already been claimed by the user, similar to how the other `lastUpdated` variables work.
Additionally, move the call to `getUserLpTokens` outside of the loop since the result is not affected by the loop iterator.
Alternatively, think about changing the implementation completely to make use of the "accumulated rewards per share" pattern (refer this blog post for a general explanation). Concretely, the implementation can update the "accumulated rewards per share" as part of the `updateAll` modifier. Then whenever a user stakes, unstakes, and transfers the DNT token, we can use the "accumulated rewards per share" to calculate and save the up-to-current user reward into storage to remove the need of running on-demand looping. For instance, `LiquidStaking.sol` can have a callback function that will be triggered when `nSBY.sol:_beforeTokenTransfer` is called. The callback function will calculate the user reward up to this point and then save the amount to storage such as a user-reward mapping. Later, `getUserRewards` can return the value from the mapping.

Update: The `getUserRewards` function now returns a stored value from `totalUserRewards[user]`.

QSP-12 Assuming Fixed Exchange Rate on the LP Token and the DNT Token

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The function `getUserLpTokens` (L188-196) assumes the exchange ratio of the LP token and the DNT token to be `mul/div`. However, if the market rate diverges from the fixed rate, it will break the calculation of `getUserRewards`.

Recommendation: The liquidity-providing platform usually provides a function to get the exchange rate for the LP token and the reserved tokens. The implementation should use those functions instead for a more accurate result. Note that depending on the DEX, this approach might be vulnerable to flash loan attacks and might require the use of an oracle instead.
Otherwise, the team should try to pick an LP token pair with stable rates and have an off-chain component to monitor. Once the rate goes off, the team should either update the `mulForLpToken` and `divForLpToken` or have a way to pause the staking rewards from LP tokens.

Update: The team removed the responsibility of calculating the token reserve amount of the LP token to separate contract(s). A new interface, `ILpHandler`, is added to call the separated contract. However, note that **the LP handler contracts are out of this audit's scope.**

QSP-13 Risk of Denial of Service with Repeated DappStaking Calls

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The function `globalWithdraw` will loop from the `lastUpdated+1` era to `_era`, which is the current era minus 1. However, the call in `DAPPS_STAKING.withdraw_unbonded()` does not need to pass in the era as an argument. In fact, `DAPPS_STAKING.withdraw_unbonded()` will withdraw from all chunks that have passed the unbond period (doc). As a result, it is redundant to loop through it. Furthermore, judging from the implementation (src), the call is likely to revert if someone has already withdrawn the funds. The redundant call will cause the `updateAll` modifier to fail and block all functions like: `stake`, `unstake`, `claim`, and `withdraw`.
A similar issue is present with `claim_dapp` as the source suggests it will return an error if rewards have been claimed already (src).

Recommendation: Remove the loop. Also, please verify whether `DAPPS_STAKING` will fail when `withdrawal_unbond` is called more than once. If it will revert, add a try-catch to prevent it from failing with the `NothingToWithdraw` error.

Update: The team has removed the `DAPPS_STAKING.withdraw_unbonded` from the loop and added a check to avoid duplicated calls. Also, the code uses a `try-catch` to prevent reversion. However, we strongly recommend catching only the specific `NothingToWithdraw` error to avoid other unexpected errors. We have a new QSP issue for this and consider this issue fixed.

QSP-14 Potentially Missing DApp Rewards

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The function `claimDapp` does not loop from the latest claimed era to the last finished era. However, the documentation of `DAPPS_STAKING.claim_dapp` states, "Claim one era of unclaimed dapp rewards for the specified contract and era". Unless someone calls the function `claimDapp` for all eras, some rewards will be lost.

Exploit Scenario: Nobody interacts with the `LiquidStaking.sol` contract for 1 era, so the `updateAll` modifier is not triggered for a specific era. Now the dapp claim for that era is lost.

Recommendation: Loop through the eras from the last claimed one in `claimDapp`.

Update: The team adds the loop to ensure calling all eras. However, the team also adds a `try-catch`. We strongly recommend catching specific errors instead of all errors. We have a new QSP issue for this and consider this one fixed.

QSP-15 `setup` Function Can Be Called More than Once

Severity: *Low Risk*

Status: Fixed

Description: Addresses with the `MANAGER` role could call `setup` method multiple times, potentially changing the `withdrawBlock` value. If changed unexpectedly it may cause certain `withdraw` transactions to fail due to the changed `withdrawBlock`.

Recommendation: Move the `setup` logic into the `initialize` function.

Update: The team has fixed the issue as recommended.

QSP-16 Duplicates Can Be Added to Arrays

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`, `contracts/nDistributor.sol`

Description: Certain arrays that are meant to keep track of sets of values can contain duplicates if purposefully added:

- `LiquidityStaking`: `stakers` array via `addStaker` (no access restriction), `lpTokens` array via `addLpToken`
- `nDistributor`: `managers` array via `addManager`, `utilityDB` and `utilities` arrays via `addUtility`, `dntDB` and `dnts` arrays via `addDnt`

Recommendation: Add the necessary validation and/or use proper enumerable set libraries to ensure that validation, addition and removal of members is efficient. OpenZeppelin's `EnumerableSet` library has a broadly used implementation of easy-to-use sets.

Update: The team has fixed the issue as recommended.

QSP-17 Use of `address.transfer`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`, `contracts/nDistributor.sol`

Related Issue(s): SWC-134

Description: The usage of `transfer` is discouraged since the callee receives a relatively low, fixed amount of gas. In case the receiver has a non-trivial fallback function, the call may always fail.

Recommendation: Use OpenZeppelin's `Address.sendValue` function or implement a similar helper function that uses the low-level `address.call` API.

Update: The team has fixed the issue as recommended.

QSP-18 Adding Staker Uses Wrong Address for Balance

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The function `addStaker` is called by the distributor to add a new staker. However the function has an issue since it updates `stakes[msg.sender]` with the balance of `_addr`.

Recommendation: Since `stakes` doesn't seem to have any utility, consider removing it altogether. If this is not a satisfying solution, fix the code by updating `stakes[_addr]` instead.

Update: The team has fixed this by updating the code to `stakes[_addr]` instead.

QSP-19 Gas Concerns: `updateAll`

Severity: *Low Risk*

Status: Fixed

Description: The `updateAll` modifier calls several functions to sync the status from the `LiquidStaking` contract with the `DAPPS_STAKING` contract. However, most synchronization relies on looping through the eras to ensure all the previous eras are updated. If no one calls the contract for a long period, the `updateAll` modifier will fail due to the loop running out of gas. The failure of `updateAll` will cause all user-facing functions to fail.

Recommendation: `lastUpdated`, `lastStaked`, and `lastUnstaked` are the three variables that are used as the start indices of the loops. We recommend the following to mitigate the issue:

1. In the `globalStake` function, set `lastStaked` to `_era` regardless of whether `sum2stake` is greater than 0.
2. In the `globalUnstake` function, set `lastUnstaked` to `_era` regardless of whether `sum2unstake` is greater than 0.
3. Add a new external `sync(era)` function. This function will perform similar calls to `updateAll`, with the difference that `era` can be specified within the range of

`(lastUpdated, currentEra() - 1]`. This function can be called periodically from an off-chain component to ensure `lastUpdated`, `lastStaked`, and `lastUnStaked` are updated. Also, if the contract gets out of sync for too long, the team can call this function to update the era to somewhere in the middle gradually and then repeat the call until all the eras get synced.

Update: The team has fixed the issue as recommended.

QSP-20 Hardcoded Initialization of `lastStaked` and `lastUnstaked`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The `initialize` function hardcodes the storage variables `lastStaked` and `lastUnstaked` value to `1175` and `1178` respectively. First, it is unclear where the values `1175` and `1178` come from. Second, this causes out-of-gas concerns for calls to `globalStake` and `globalUnstake` as those functions contain loops from `lastStaked` and `lastUnstaked` variables. Once out of gas, it will cause the `updateAll` modifier to fail, which breaks all important user-facing functions.

Recommendation: Use the `currentEra()` function to initialize the variables `lastStaked` and `lastUnstaked`.

Update: The team has fixed the issue as recommended.

QSP-21 Risk of Missing Variable Initialization

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: Some setter functions, such as `setup`, `setDistr`, and `setDntToken`, need to be called before the contract can properly function. If the deployment scripts/process misses calling those setter functions, the contract will be unusable.

Recommendation: The following variables should be set in `initialize`: `withdrawBlock`, `DNTname`, `utilName`, `distrAddr`, `distr`, and `dntToken`.

Update: The team has fixed the issue as recommended.

QSP-22 Potential Balance Inconsistency

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/nDistributor.sol`, `contracts/nSBY.sol`

Description: The function `nDistributor.sol:assignUtilityFromNull` (L525-535) will move the "null" utility amount to the `_newUtility`. If the `MANAGER` role ever calls the function with `_newUtility` equal to "LiquidStaking", it will add the "null" utility amount to the "LiquidStaking" utility. This will cause the balance inside the `nSBY` token to be inconsistent with the utility amount inside the `nDistributor` contract and break the consistency mechanism used in `nSBY.sol:_beforeTokenTransfer` that hooks `nDistributor.sol:transferDnt` to keep the two consistent.

Recommendation: The team should consider having a disallow list for`_newUtility` when calling the `assignUtilityFromNull` function. Alternatively, the team should be aware of this and never call `assignUtilityFromNull` with "LiquidStaking".

Update: The team has fixed the issue by introducing `disallowList`.

QSP-23 Gas Concerns: `nDistributor.sol`

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `contracts/nDistributor.sol`

Description: There are some function in the `nDistributor.sol` contract that loop through arrays from storage, namely `_addDntToUser`, `_addUtilityToUser`, `_removeUtilityFromUser` and `_removeDntFromUser`. Those loops could be problematic once the list of the user's DNTs or utilities grows too big. However, this is less of an issue during the start as the team plans to start with one DNT and one utility only.

Recommendation: The team should be aware of this and not provide too many DNTs and utilities. Also, we recommend enhancing the implementation regarding array element deletion. Instead of `delete <:your_array>[i]`, a better implementation is to swap the last item to the targeted index to delete and then delete the value on the last index. Otherwise, the current implementation will leave an empty value in the array, but the index of the array will continue to grow despite the deletion. The deletion occurs at `removeManager` (L195), `_removeUtilityFromUser` (L464), and `_removeDntFromUser` (L481).

Update: The team has improved the deletion logic. Note the gas issue still exists, and the team should be aware of this and not provide too many DNTs and utilities.

QSP-24 Risk of Issuing Wrong Utilities on Transfer

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/nDistributor.sol`

Description: On L517 inside the function `transferDnt`, it hardcodes the utility as "LiquidStaking" when calling `issueTransferDnt`. If the upstream function calls `transferDnt` with a different `_utility`, this will issue the wrong DNT to the user.

Recommendation: Use the variable `_utility` instead of the hardcoded "LiquidStaking" string.

Update: The team has fixed the issue as recommended.

QSP-25 Lack of Events on Critical State Changes

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`, `contracts/nDistributor.sol`

Description: Both the `LiquidStaking` and `nDistributor` contracts lack any kind of events beyond the ones emitted by the used library contracts. Events are important as they allow state changes to easily and efficiently be tracked externally.

Recommendation: Define and emit the necessary events.

Update: The team has added events (`Staked`, `Unstaked`, `Withdrawn`, and `Claimed`) to the `LiquidStaking.sol` contract.

QSP-26 Unlocked Pragma

Severity: *Informational*

Status: Mitigated

File(s) affected: `contracts/LiquidStaking.sol`, `contracts/nDistributor.sol`, `contracts/nSBY.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. 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, we recommend to remove the caret to lock the file onto a specific Solidity version.

Update: The team locks the `LiquidStaking.sol` and `nDistributor.sol` to `0.8.4`. However, the `nSBY.sol` file has not been changed.

QSP-27 Unused or Unnecessary Variables and Functions

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description:
Several variables and functions seem unused or unnecessary. They make the code harder to read and follow.

Variables:

- `LiquidStaking.sol:L80`: The `shadowTokensAmount` mapping.
- `LiquidStaking.sol:L85`: The `hasLpTokens` mapping is only used in the function `addToLpOwners`. The function will set entries of the `hasLpTokens` mapping to `true`, but none of the code reads the `hasLpTokens` mapping.
- `LiquidStaking.sol:L87`: The `lastRewardsCalculated` variable. Although the manager can set this value in `setLRC`, none of the code uses this variable. It is only used in commented out code on `L198-222`.
- `LiquidStaking.sol:L72`: The `rewardsByAddress` mapping. The code will set the mapping of a certain address to 0 at `LiquidStaking.sol:addStakers`. However, it is never read or updated. All code related to this variable has been commented out.

Functions and modifiers:

- `LiquidStaking.sol:addToLpOwners`: This function sets entries of the `hasLpTokens` mapping to true, but the mapping is never read.
- `LiquidStaking.sol:setLRC`: Despite the manager being able to call this function to set values in `lastRewardsCalculated`, none of the code uses the variable `lastRewardsCalculated`. It is only used in commented out code on `L198-222`.
- `LiquidStaking.sol:chck`: Empty function.
- `LiquidStaking.sol:calcReward`: Empty modifier.

Recommendation: Remove the variables and functions listed above.

Update: The team has removed all unused or unnecessary variables and functions as pointed out in the original issue.

QSP-28 Role Change without Manager/Owner Update

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `contracts/nDistributor.sol`

Description: The `nDisitributor` contract has certain methods that besides changing its own direct state also change roles, specifically `changeOwner` and `addManager`. These not only update the associated state `owner` and `managers` respectively but also assign and/or revoke the roles from the appropriate addresses. However the used `AccessControlUpgradeable` library allows for the assignment and revocation of roles without affecting the additional state variables. The `owner` and `managers` variables don't seem to be used anywhere in the contract's logic.

Recommendation: Clarify whether the `owner` and `managers` state variables should always be consistent with the roles or whether the `changeOwner` and `addManager` methods are simply for consistence.

Update: The team acknowledged this with the following statement.

> There is a chance that the contents of managers and owner will not correspond with the reality, if roles are changed directly through the AccessControl. However, this can only be done by a manager, which could only be changed through the added functions, therefore, we don't consider this to be a vulnerability and have kept everything as is

QSP-29 Lack of Ability to Remove LP Tokens

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The `LiquidStaking` contract has a method for adding LP tokens `addLpToken`, however there is no method for removing tokens.

Recommendation: Clarify whether the inability to remove LP tokens is intentional and adjust the contract's logic accordingly.

Update: The team has added the `removeLpToken` function.

QSP-30 Unclear Revenue Accounting

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The `LiquidStaking` contract tracks an `eraRevenue` variable which collects 1% (L330) of accrued staker rewards and a 1% staking fee (L440). However 1/10th of the revenue is added to the `unstakingPool` (L506). Not only is it unclear what the contract is meant to do with the remaining 90% of the revenue but the taken 10% is not deducted from the respective `eraRevenue`.

Recommendation: Clarify whether owners are meant to somehow be able to withdraw revenue or generally what is meant to happen to it. Furthermore, clarify how the contract is intended to track revenue and whether or not the `eraRevenue` variable should be adjusted when revenue is taken.

Update: The team adds a `withdrawRevenue` function to allow the admin to withdraw the revenue. Also, the team adds the variable `totalRevenue` to track.

QSP-31 Lacking Access Control

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The `addToLpOwners` method lacks any validation. It does not check the caller or whether the specified `_user` can or should be set to `true`. There is generally little documentation and a complete lack of specification specifying what exactly the `addToLpOwners` method, and `hasLpTokens` mapping is meant to do.

Recommendation: Clarify the expected behavior and adjust the code accordingly.

Update: The team removes the function `addToLpOwners`.

QSP-32 Limited Integrity Of Code Supply Chain

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The OpenZeppelin smart contract library is directly copied into the repository, rather than being included with either a package manager (npm, yarn) or version control system (git submodule). This is problematic as it can make it difficult to tell how correct and/or safe the libraries are as errors may have been made during the initial copying process. Furthermore, the upgradeable and non-upgradeable library versions seem to mixed together with some folders containing both contract versions and other not.

Recommendation: It is recommended that smart contract dependencies are handled properly in a code repository to ensure their integrity. A mistake in one line or even a couple characters may lead to severe unforeseen consequences.

Update: All dependencies have been updated.

QSP-33 Problems with New Reward Computation

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: The re-audit commit `15f9daf8f` re-designs the logic of reward computation. The new design will have an off-chain script looping through all users to record the share of a specific user at a specific time of an era. In the next era, it uses the medium value to calculate the reward of the previous era for fairness. The new design aims to mitigate the issues listed in the `QSP-7 Problems With Reward Computation`. However, we notice some issues in `LiquidStaking.sol:eraShot`:

1. `nTotal` is not the sum of user balances. `nTotal` is set as `distr.getUserDntBalanceInUtil(_user, _util, _dnt)` . Judging from the implementation, the `dntInUtil` will mostly be the same as the token balance and not the sum of balances from all users. The gap will cause the calculation in L226 on the `nShare` variable to be wrong.
2. The calculation of `nShare` can go underflow. The `buffer` is added when a user receives the token. However, if the user transfers away those tokens after receiving them, the `nBal` can go lower than the `buffer`. `(nBal + lpBal - buffer[_user][era])` can go underflow and revert.
3. `nShare` is quite likely to be zero, always. The `nBal + lpBal - buffer` is supposed to be less than `nTotal` as this calculates the share percentage. However, if not multiplied with some values (e.g., 10 ^ precision_digits), the division will result in zero.
4. gas concern when calling `findMean` in L216. The complexity of `findMean` is `O(n^2)`.
5. `algemEraStaked` and `totalEraStaked` (L223-234) should be applied together with the `prevEraRewards` (L214). The idea is to calculate the reward of this `LiquidStaking.sol` contract and then distribute the reward of the `LiquidityStaking` contract to each user according to the shares. The number of `algemEraStaked` and `totalEraStaked` is likely to differ from the final number before the era ends.

Recommendation: We recommend fixing the following:

1. Use the `totalBalance` of the DNT token instead for the `nTotal`.
2. Either view the share as 0 when there is more buffer than the balance or decrease the buffer on transferring out to avoid the underflow.
3. Multiple with `10^precision_digits` first when calculating `nShare`.
4. Add a cap on the length of the `userIncompleteEraRewards[_user][era]` array. Revert when the length becomes impractical.
5. Consider calling `DAPPS_STAKING.claim_dapp` here and using the balance difference as the total reward. This removes the risk of calculating from `DAPPS_STAKING.read_era_reward`, `DAPPS_STAKING.read_staked_amount_on_contract`, and `DAPPS_STAKING.read_era_staked` to calculate the reward of specific era. Calling `DAPPS_STAKING.claim_dapp` here also removes the need for the `claimDapp` function.

Update: The team fixed the issue in the re-audit commit `b11a660`.

1. The team fixed the `nTotal` with `distr.totalDntInUtil(_util)` instead. The `nDistributor.sol` contract will record the total amount of tokens for the utility.
2. The team adds a new function `LiquidStaking.sol:setBuffer(...)` and the `nDistributor.sol:transferDnt` (L642) uses that to cap the buffer for the sender.
3. The team fixed as recommended.
4. The team fixed as recommended.
5. There is a mistake in the recommendation. The code should call `DAPPS_STAKING.claim_staker` for the reward. The `DAPPS_STAKING.claim_dapp` is not intended to share with the staker. The team fixed the issue as recommended but used `DAPPS_STAKING.claim_staker` instead.

QSP-34 Risk of the `sync` Function

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: In the re-audit commit `15f9daf8f`, the team added the `sync` function to allow the possibility to sync to an intermediate era if gas is not enough to sync all the way to the current era. However, we noticed a few issues:

1. The function does not validate the `_era` input. The `_era` should only be in the range of `(lastUpdated, currentEra() - 1]`
2. `lastUpdated` is not recorded after syncing.

Exploit Scenario: The manager accidentally calls the `sync` function with an old era. The miscall could overwrite the `lastStaked` and `lastUnstaked` to a previous era in the `globalStake` and the `globalUnstake` functions.

Recommendation: We recommends the following:

1. Validate the `_era` is in the range of `(lastUpdated, currentEra() - 1]`.
2. Update the `lastUpdated` to the `_era`.
3. Add a NatSpec on the function to explain the reason and the usage of this function.
4. (optional) Refactor the `updateAll` to call `sync` to reuse the code inside the `if (lastUpdated != era)` block.

Update: The team fixed as recommended in the re-audit commit `b11a660`. Note that the function `globalUnstake` has been refactored and no longer loops through eras.

QSP-35 Potential Inconsistency on `lphandlers` and `lptokens`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/LiquidStaking.sol`

Description: In the re-audit commit `15f9daf8f`, the team added a new mapping `lpHandlers` to remove the responsibility of the LP reserve calculation from the `LiquidStaking.sol` contract. The function `addLpHandler` can set the `lpHandlers`. However, there is another function, `addLpToken`, to set the array of the `lpTokens`, but the `addLpHandler` does not check whether the LP token exists inside the `lpTokens` array. If they are not set together, the function `getUserLpTokens` could fail unexpectedly.
Note that a similar inconsistency occurs on the `removeLpToken` function. The function did not remove the `lpHandlers`.

Exploit Scenario: The manager sets the `lpTokens` but forgets to add the LP handler. Now, the manager cannot call `eraShot` because the `getUserLpTokens` function will fail when calling `ILpHandler(lpHandlers[_lpTokens[i]]).calc(_user)`.

Recommendation: It seems unnecessary to have two functions, `addLpToken` and `addLpHandler`. We recommend merging the two and setting all LP-related data together. Note that the function `removeLpToken` should erase the data consistently too.

Update: In the re-audit commit `b11a660`, the functions `addLpToken` and `addLpHandler` now becomes a new function `addPartner`. Also, the function `removeLpToken` resets the `lpHandlers` mapping.

QSP-36 Catching All Exceptions

Severity: *Undetermined*

Status: Mitigated

File(s) affected: `contracts/LiquidStaking.sol`

Description: In the re-audit commit `15f9daf8f`, the team added the `try-catch` when interacting with the `DappsStaking` to avoid specific failures reported in the issue. However, the implementation adds a `try-catch` that mutes all exceptions. The code should only prevent the excepted ones from failing instead of catching everything.

Exploit Scenario: If the Algem is not registered, it is supposed to fail as the contract is not activated on the network (code). However, it is supposed to block our contract from proceeding with the era, but the current implementation will mute the error.

Recommendation: We recommend doing the followings:

1. Only mute the exception on specific known errors. Revert on those unexpected ones.
2. Consider logging an event on the exception message when muting them.

Update: In the re-audit commit `b11a660`, the team fixed some places, remaining the following calls open to error. The team states that they plan to monitor the events on error to gain operational experience and fix later. - `DAPPS_STAKING.claim_staker`: this one seems less concerned. The worse case is missing the reward for certain era, but the system can continue to work. - `DAPPS_STAKING.unbond_and_unstake`: once failed, there will be not enough balance to withdraw. The operation team should monitor the `UpdateError` event and call the function `fillUnbonded` to avoid failure of withdrawal.

Automated Analyses

Slither

Slither report 302 results. These results were either related to code from dependencies, false positives or have been integrated in the findings or best practices of this report.

Code Documentation

1. Add a comment on `LiquidStaking.sol:fillPools` to indicate that this function is expected to always run together with and after the `LiquidityStaking.sol:globalClaim` function with the same `_era` parameter. Otherwise, the implementation regarding `eraStakerReward` will be incorrect.
2. [fixed] Add a comment in `nDistributor.sol:initialize` (L152-162) to specify the intention of the "empty" utility. It seems to be a dummy utility to bump the `utilityId` and `dntId` to start from 1 instead of 0.
3. Add a sample use case of the `nDistributor.sol:assignUtilityFromNull` function to demonstrate how a user will use this function in the documentation.
4. Add a comment that `utilities` will exclude the "empty" utility, and the index will differ from the one in `utilityDB`.
5. (re-audit commit `b11a660`) The comments in `LiquidStaking.sol:L227-229` is out-dated. In L230 it uses the condition of `lastClaimed != era` and not checking if `_user` is first staker in the list anymore.

Adherence to Best Practices

1. [fixed] `contracts/nDistributor.sol`: L599: Duplicate check. The check on L599 is already done in the called `dntInterface` modifier. Checking again uses unnecessary gas.
2. `(all)`: Lack of code linter and/or fixer. It is recommended a linter is used to ensure consistent indentation and syntax structure.
3. [fixed] Using two variables for the distributor in `LiquidStaking` is redundant. Use a single variable of type `NDistributor` and cast it to an address if needed like this `address(distr)`.
4. [ack] `NSBY` inherits from `Ownable` and `AccessControl`. Since `AccessControl` provides a superset of the features of `Ownable`, consider using just `AccessControl`. **Update:** the team responded that "We have decided to keep Ownable for the comfort and simplicity".
5. [mitigated] Almost all functions are declared as `public`, even if they are not called from within the contract. Declare functions as `external` by default instead. **Update:** the file `LiquidStaking.sol` has fixed most of this with a leftover on the function `getLpTokens`. However, other contracts: `nSBY.sol` and `nDistributor.sol` has not been not updated.
6. Some contract variables use the default visibility. Consider explicitly declaring all visibilities to avoid potentially surprising defaults.
7. [fixed] Consider removing access control logic from view functions since everything on chain is public anyway and can be retrieved with web3 API calls (see `LiquidStaking.getStakers`).
8. [fixed] Remove commented out code:
    - `nDistributor.sol:L445-451`: inside the function `removeTransferDnt`
    - `LiquidStaking.sol:L192-222`: The commented function `user_reward`
    - `LiquidStaking.sol:L339-343,L345-350`: Inside the function `claimDapp`
    - `LiquidStaking.sol:L372-378`: Inside the modifier `calcReward`
    - `LiquidStaking.sol:L462`: Inside the function `claim`
9. [fixed] The re-audit commit `15f9daf8f` introduces the `LiquidStaking.sol:findMean` function as part of the new design to calculate the reward. There is space for improvement on the function itself:
    1. The function `findMean` naming should be `findMedium`. The function returns the medium instead of the average.
    2. In L177, the `uMax` can be set as `type(uint).max` instead of the hardcoded value for better readability. Otherwise, please provide a comment describing the value itself.
    3. In L183, the `min` should be set to `type(uint).max` without `-1`. Though unlikely to happen, if the array has the value of `type(uint).max`, this function will return a wrong value.
    4. The implementation could use the bubble sort algorithm to save some gas costs without complicating the code. In a typical bubble sort, it loops `(n^2) / 2` times instead of the current implementation of `n * (2 * n)`. The main difference is to swap the value instead of forming a new array. See example: link.
10. [fixed] Please remove the commented code in `nDistributor.sol:L378-380` (inside the `issueTransferDnt` function).
11. (re-audit commit `b11a660`) In `LiquidStaking.sol:L380-383` (inside `stake(...)` function), instead of try-catch + revert, a simple call without try-catch will do the same since the current implementation reverts on all cases.

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

```
c1ecdd3b94b20dc31e33e0ddf6169cdc05c4d62eb2a1cd52a85a35c4a2a08190 ./contracts/nSBY.sol

275e6cc54683522667bcbc2c66324531bf66a4a7f9362ba966e64c291d8536c0 ./contracts/LiquidStaking.sol

68f4c428803d674ae29c1b82acf7a5f8b928335e7810949ae1d8d99c9d80f6f3 ./contracts/nDistributor.sol

edbd2216991404e3bd0d77c9bbd9b60e1b04ebf879e35ffcb15093be75f0ba90 ./contracts/interfaces/IDNT.sol

dcb7b31c750f3897b56adf03146a40ee905bbe7e2613992f1700c61dbf70761a ./contracts/interfaces/DappsStaking.sol
```

Changelog

- 2022-06-24 - Initial report
- 2022-07-12 - re-audit report
- 2022-07-22 - re-audit 2 / final 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.