Quantstamp Security Assessment Certificate

May 6th 2022 — Quantstamp Verified

Blockdaemon + StakeWise

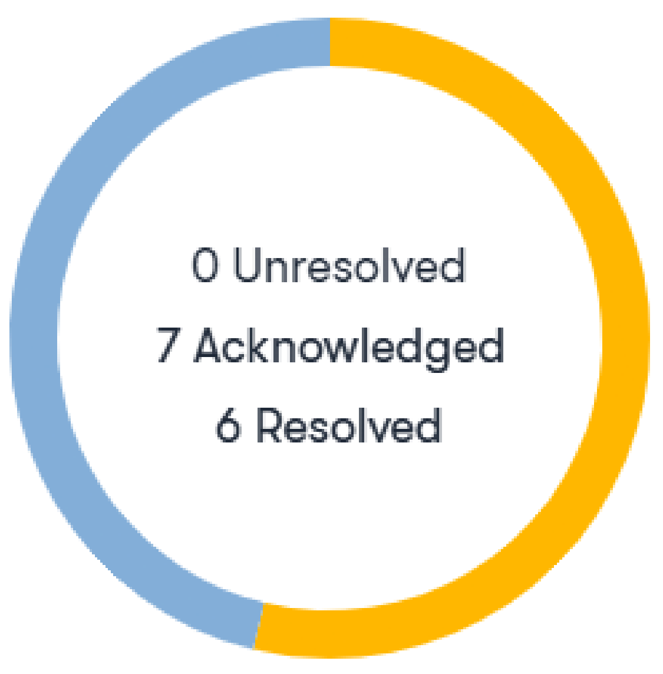
This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Liquid staking protocol
Auditors	Joseph Xu, Technical R&D Advisor Cristiano Silva, Research Engineer Sebastian Banescu, Senior Research Engineer
Timeline	2022-04-11 through 2022-05-06
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Introduction to StakeWise Smart Contracts (Permissioned)
Documentation Quality	High
Test Quality	High
Source Code	

Repository	Commit
contracts (Initial audit)	6ebf119
contracts (Re-audit 1)	5308be5
contracts (Re-audit 2)	b1e838f

Total Issues	13 (6 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	5 (3 Resolved)
Low Risk Issues	2 (0 Resolved)
Informational Risk Issues	6 (3 Resolved)
Undetermined Risk Issues	0 (0 Resolved)



Documentation Issues Addressed

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

Re-audit 2 (commit `b1e838f`): Quantstamp has performed a second re-audit of the StakeWise smart contracts. The team has fixed and mitigated the two outstanding "Medium" severity issues surrounding the whitelisting feature from the first re-audit. The team has provided additional details and acknowledged the two unresolved "Informational" severity issues from the first re-audit. Recommendations in "Code Documentation" and "Adherence to Best Practices" sections were addressed. There are no failed tests and the auditors were able to obtain code coverage data.

In particular, the team has provided a detailed documentation surrounding the permissioned version of the protocol, which is available at https://docs.stakewise.io/smart-contracts-permissioned. This documentation clearly indicates contract functionalities, on-vs-off-chain components, and user interactions (including a high-level architecture diagram).

Re-audit 1 (commit `5308be5`): Quantstamp has performed a re-audit of the StakeWise smart contracts. The team has mitigated one of the "Medium" severity issues surrounding the whitelisting feature. However, the issue of detailed specification or documentation remains unresolved. The changes to `StakedEthToken` appear to introduce a new risk (possibly stuck funds). Another issue related to the accrual of `RewardEther` token to non-whitelisted accounts is acknowledged. Of the remaining issues not related to the whitelisting feature, three "Informational" severity issues are fixed or mitigated. Four "Medium" to "Informational" severity issues are acknowledged (one "Medium", two "Low", and one "Informational" severity), and two "Informational" severity issues remain unresolved. The team did not address the recommendations in the "Code Documentation" or the "Adherence to Best Practices" sections.

Initial audit (commit `6ebf119`): Quantstamp has performed an audit of the StakeWise smart contracts intended for institutional liquid staking in partnership with Blockdaemon. The protocol consists of on-chain smart contracts and off-chain programs that help manage oracle updates, validator registration, and rewards distribution. This audit's scope includes only the on-chain smart contracts. None of the off-chain programs were considered as part of this audit.

Quantstamp has identified 13 issues in total, with 5 marked as "Medium" severity. The "Medium" severity issues are mostly related to the lack of specification of new whitelisting feature that was added to the StakeWise protocol, which has lead to several edge cases not being considered. One "Medium" severity issue is related to the level of privilege involved in setting the protocol fee. In addition, there are 2 issues marked as "Low", 6 issues marked as "Informational" severity, and 3 "Best Practices" recommendations.

Quantstamp strongly recommends addressing all findings, including the best practice issues mentioned in this report. In particular, the auditors recommend developing a more detailed specification for the whitelisting feature.

ID	Description	Severity	Status
QSP-1	Lack of specifications and documentations for the whitelist functionality	^ Medium	Fixed
QSP-2	Non-whitelisted accounts may mint and transfer `StakedEthToken`	^ Medium	Mitigated
QSP-3	`RewardEthToken` can accrue to non-whitelisted accounts	^ Medium	Acknowledged
QSP-4	Partner and referrer addresses provided on-chain may not be on the whitelist	^ Medium	Mitigated
QSP-5	High upper bound for the protocol fee	^ Medium	Acknowledged
QSP-6	Operator data is not well-preserved	⌄ Low	Acknowledged
QSP-7	Implementation of withdrawal from ETH2 is unclear	⌄ Low	Acknowledged
QSP-8	Privileged roles and ownership	○ Informational	Acknowledged
QSP-9	Violation of the "Checks-Effects-Interactions" pattern	○ Informational	Acknowledged
QSP-10	Lack of input sanitization	○ Informational	Fixed
QSP-11	Gas usage/for-loop concerns	○ Informational	Fixed
QSP-12	Front-running of the `initialize()` function	○ Informational	Mitigated
QSP-13	Clone-and-own	○ Informational	Acknowledged

Quantstamp Audit Breakdown

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

DISCLAIMER: This protocol contains components that rely on off-chain programs. These off-chain programs play important roles in functionalities such as oracle updates, validator registration, and reward distribution when special types of addresses (operator, partner, or referrer) are involved. The off-chain programs were not considered as part of this 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 v0.8.2
- Mythril 0.22.41

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run the Slither tool from the project directory: `slither .`
3. Install the Mythril tool: `pip3 install mythril`
4. Run the Mythril tool on each contract: `myth analyze <path/to/contract>`

Findings

QSP-1 Lack of specifications and documentations for the whitelist functionality

Severity: *Medium Risk*

Status: Fixed

Description: The main changes in this commit of the StakeWise smart contracts from those currently deployed on the Ethereum Mainnet is the addition of the whitelist feature using the `WhiteListManager.sol` contract. However, specifications of the whitelist feature is not provided and there is no public documentation corresponding to this functionality either. The lack of specification and documentation leads to several edge cases not being handled well (and potentially other issues being present), as is discussed in subsequent issues QSP-2, 3, and 4.

Recommendation: We recommend developing a detailed specification for the whitelisting functionality. Information such as which actions can be performed by whitelisted users, which actions can be performed by non-whitelisted users, and edge cases such as what happens if a user moves from whitelisted to non-whitelisted, and vice-versa should be clearly answered.

Update: [Re-audit 2] The team has provided a detailed specification for the permissioned version of the protocol at https://docs.stakewise.io/smart-contracts-permissioned. This documentation clearly indicates contract functionalities, on-vs-off-chain components, and user interactions (including a high-level architecture diagram).

[Re-audit 1] The team has provided a high level description of the functionality: "Whitelisted accounts should be able to transfer and mint tokens (it's also stated in the `WhiteListManager` contract), not whitelisted accounts should not be able to interact with the protocol. In case the account was removed from the whitelist, the permissioned assets can remain in its wallet, but there should be no way to transfer them to anyone else."

QSP-2 Non-whitelisted accounts may mint and transfer `StakedEthToken`

Severity: Medium Risk

Status: Mitigated

File(s) affected: `./tokens/StakedEthToken.sol`

Description: The `mint()` function in `StakedEthToken` does not check if the recipient is whitelisted. This means that a non-whitelisted account may obtain `StakedEthToken`. Such situation may arise if a whitelisted account is de-whitelisted while waiting for validator activation after depositing a large amount of ETH to the pool contract.

Similarly, the `_transfer()` function in `StakedEthToken` does not check if the sender is whitelisted. This means that a non-whitelisted account may transfer `StakedEthToken` to a whitelisted address and corrupt the on-chain record (e.g., if the non-whitelisted account was de-whitelisted due to sanctions).

Recommendation: It is unclear whether this is intentional or not, as the whitelisting specification was not provided. Clarify whether or not non-whitelisted accounts are never supposed to receive mint or transfer out their `StakedEthToken`. Note that in such case, the privilege of the admin account will increase further because it can freeze the `StakedEthToken` of an address by de-whitelisting.

Update: [Re-audit 2] The fix in PR-2 has been reverted. The team acknowledges that the edge case discussed in this report (i.e., an address is de-whitelisted before `stakedETH` is minted through `Pool.activate()`) falls under a special case outlined in the high level description "In case the account was removed from the whitelist, the permissioned assets can remain in its wallet, but there should be no way to transfer them to anyone else." Therefore, non-whitelisted accounts may mint tokens but may not transfer tokens.

[Re-audit 1] The changes in PR-2 result in non-whitelisted accounts no longer being able to mint `stakedETH` or transfer tokens (both `stakedETH` and `rewardETH`). However, it is unclear what should happen to the ETH deposit for a previously whitelisted account who is waiting to have `stakedETH` minted due to a large ETH deposit who then gets de-whitelisted before `stakedETH` can be minted using the `Pool.activate()` function. Note that this account would at least be able to mint `stakedETH` before the change in PR-2 but now this account would have the funds stuck because the account now cannot mint `stakedETH` or withdraw the ETH deposit.

As a result, this specific edge case would be inconsistent with the high-level description provided by the team in response to QSP-1: "In case the account was removed from the whitelist, the permissioned assets can remain in its wallet, but there should be no way to transfer them to anyone else."

QSP-3 `RewardEthToken` can accrue to non-whitelisted accounts

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `./tokens/RewardEthToken.sol`

Description: A non-whitelisted account will continue to accrue `RewardEthToken` as long as it holds `StakedEthToken`. Such situation may arise if a whitelisted account holding `StakedEthToken` is de-whitelisted.

Recommendation: It is unclear whether this is intentional or not, as the whitelisting specification was not provided. Clarify whether or not a non-whitelisted account may continue to accrue rewards. Note that in the latter case, the privilege of the admin account will increase further.

Update: From the team: "In case the account was removed from the whitelist, any interactions with `prETH`, `psETH` shouldn't be possible However, disabling rewards accrual to such an account would complicate the rebalancing logic in the `RewardEthToken` contract. As of now, it's acceptable for rewards to accrue as long as they cannot be transferred out."

QSP-4 Partner and referrer addresses provided on-chain may not be on the whitelist

Severity: Medium Risk

Status: Mitigated

File(s) affected: `./pool/Pool.sol`

Description: The `Pool` contract provides functions that allow deposits to the protocol through a partner or referrer address (`stakeWithPartner()`, `stakeWithPartnerOnBehalf()`, `stakeWithReferrer()`, `stakeWithReferrerOnBehalf()` respectively). These functions process deposits in the same manner as the normal `stake()` and `stakeOnBehalf()` functions, but emit events containing the partner and referrer addresses, which are then processed off-chain. While the intention seems to be that partner and referrer addresses must be whitelisted, it is possible to submit on-chain partner or referrer addresses that are non-whitelisted, which would then get processed in the same way as a regular deposit.

Recommendation: Clarify how the information on partner or referrer addresses are processed off-chain, and what would happen if they are not whitelisted. Alternatively, add a check to enforce that partner and referrer address are already whitelisted.

Update: From the team: "Adding check for partners and referrers to be whitelisted would cost additional gas to the user. The checks are done off-chain when the reward are transferred to them. Even if `prETH` will be allocated to them (it's done through the `MerkelDistributor`), they won't be able to claim it (see `RewardEthToken.claim()`)."

QSP-5 High upper bound for the protocol fee

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `./tokens/RewardEthToken.sol`

Description: The protocol fee's may be set as high as 99.99% based on the current check in the `RewardEthToken.setProtocolFee()` function. Users must trust the protocol operator to not take advantage of this fact, especially after the liquid staking pool attracts significant number of deposits.

Recommendation: Modify the protocol fee upper bound to a more reasonable value, or communicate the current upper bound explicitly to the users.

Update: From the team: "As it's currently not clear what would be the APR post merge, it's better to not set upper bound on the fee. Later, we will either remove the ability to change it or upgrade the contract with upper bound in place."

QSP-6 Operator data is not well-preserved

Severity: Low Risk

Status: Acknowledged

File(s) affected: `./pool/PoolValidators.sol`

Description: The data stored in the mapping `PoolValidators.operators` can easily be overwritten or deleted, even if the operator status is commited or if the operator has had a validator registered. The former case (overwrite) may happen if `PoolValidators.addOperator()` is called with a new `depositDataMerkleRoot` and the latter case (deletion) may happen simply by calling `PoolValidators.removeOperator()` function.

Recommendation: Clarify the workflow around operator addition, commit, and removal. Indicate whether the operator data is intended to be a temporary record within this workflow or a permanent record.

Update: From the team: "That's intentional. Once the operator runs out of keys, he will generate a new `depositDataMerkleRoot` that will override the current value."

QSP-7 Implementation of withdrawal from ETH2 is unclear

Severity: Low Risk

Status: Acknowledged

File(s) affected: `./pool/PoolEscrow.sol`

Description: The repository currently has a contract called `PoolEscrow`, which will receive the rewards from ETH2 after withdrawals are live. However, the exact implementation details of how the rewards will be withdrawn and distributed is unclear. As of now, the current implementation is permissioned and withdrawals/claims will be conducted at the discretion of the `PoolEscrow` contract owner.

In addition, care must be taken to trust the payee or to avoid reentrancy vulnerabilities. The Stakewise team appears to be aware of this issue and has indicated this in the code comments of `PoolEscrow.withdraw()` function.

Recommendation: Clarify any specification or implementation plans that are currently in place for withdrawing and distributing ETH2 rewards. In addition, make sure to use safeguards such as the "Checks-Effects-Interactions" pattern or the `ReentrancyGuard` contract when implementing the withdrawal function.

Update: From the team: "We've decided to postpone implementation of the withdrawals until the ETH2 post-merge fork. We want to be able to perform end-to-end withdrawal tests before enabling that for users. The plan is to trigger validator withdrawal, burn `prETH`, `psETH` and return ETH to the user."

QSP-8 Privileged roles and ownership

Severity: Informational

Status: Acknowledged

File(s) affected: `./pool/PoolValidators.sol`, `./tokens/RewardEthToken.sol`, `./tokens/StakedEthToken.sol`, `./WhiteListManager.sol`

Description: This repository contains the contracts for a permissioned version of the Stakewise liquid staking protocol. While the permissioned nature is intended, the protocol admin nevertheless has strong privileges over users including:

- Account whitelisting (with unclear consequences)
- Pausing and unpausing of token contracts
- Updates of protocol parameters
- Management of the operator data
- Management of eventual ETH2 reward distribution
- Management of off-chain components.

Recommendation: Clarify the extent of the administrative privileges to users in public documentation.

Update: From the team: "The protocol admin role will be assigned to the multisig that is controlled by StakeWise, BlockDaemon, and some other independent parties. Once protocol matures and there will be no need to upgrade contracts, or tweak parameters, we will reduce the number of admin functions."

QSP-9 Violation of the "Checks-Effects-Interactions" pattern

Severity: Informational

Status: Acknowledged

File(s) affected: `./tokens/RewardEthToken.sol`, `./tokens/StakedEthToken.sol`

Description: Critical functions such as `StakedEthToken._transfer()`, `StakedEthToken.mint()`, and `StakedEthToken.toggleRewards()` violate the "Checks-Effects-Interactions" pattern by updating its state variables after an external call to the `RewardEthToken` contract.

Recommendation: While the current implementation does not lead to a vulnerability, we recommend checking the `RewardEthToken.rewardDisabled[address]` flag independently of the reward update, as is currently done in the function `RewardEthToken.updateRewardCheckpoint()`. This will allow the `RewardEthToken` reward update call to take place at the end of the functions in question.

Update: [Re-audit 2] The auditors have recommended the team to address this issue, which was marked as "Unresolved" during the first re-audit. The team has responded as follows: "[addressing this issue] will cause additional gas cost (1 extra storage read for `StakedEthToken.mint` and 2 extra storage reads for `StakedEthToken.transfer`). These two functions are the most used ones, so we want to optimize there as much as we can. Also, I can't think of any case when we will add an external contract call that will introduce re-entrancy there. It's very unlikely that we will update this functionality in future. Also, this logic was in prod [auditor note: in the permissionless version of the protocol] for close to a year now, so we want to keep chances of introducing new issues minimal."

[Re-audit 1] From the team: "We've acknowledged that recommendation but won't apply it now as it will introduce an additional gas cost."

QSP-10 Lack of input sanitization

Severity: Informational

Status: Fixed

File(s) affected: `./pool/PoolEscrow.sol`, `./Oracles.sol`

Description: The following functions do not validate input addresses from being the zero address:

- `PoolEscrow.constructor()`
- `Oracles.addOracle()`

Recommendation: Unless input addresses are known to be correct (e.g., controlled by deployment scripts), we suggest verifying that they are different than `address(0)` or they are indeed contracts (when applicable).

Update: Fixed in PR-2.

QSP-11 Gas usage/for-loop concerns

Severity: Informational

Status: Fixed

File(s) affected: `./Oracles.sol`

Description: The following functions in `Oracles.sol` present nested loops. Depending on the input, such as an unusually high number of signatures submitted in the `signatures` array, the function execution can run out of gas:

- `Oracles.submitReward()`
- `Oracles.submitMerkleRoot()`
- `Oracles.registerValidator()`

Recommendation: In addition to checking for quorum through `Oracles.isEnoughSignatures()`, check that the length of signatures array do not exceed the number of oracles.

Update: Fixed in PR-2.

QSP-12 Front-running of the `initialize()` function

Severity: Informational

Status: Mitigated

Description: The `initialize()` function used to initialize important contract states can be called by anyone. An attacker can initialize the contract before the legitimate deployer, hoping that the victim continues to use the same contract. If the `initialize()` function is used to set the contract owner, then attackers can potentially set themselves as the owner of the contract by frontrunning the `initialize()` function call. In the best case, the victim can notice this attack but must redeploy the contract for additional gas cost.

Recommendation: Use the constructor to initialize non-proxied contracts. To initialize proxy contracts, deploy contracts using a factory contract that immediately calls `initialize()` after deployment, or make sure to call `initialize()` immediately after deployment and verify a successful transaction.

Update: From the team: "We call initialize immediately after the deployment using an automated script. After all the contracts are deployed, we run tests on mainnet fork and verify the initialized state."

QSP-13 Clone-and-own

Severity: Informational

Status: Acknowledged

File(s) affected: `./presets/OwnablePausable.sol`, `./presets/OwnablePausableUpgradeable.sol`, `./tokens/ERC20PermitUpgradeable.sol`, `./tokens/ERC20Upgradeable.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. The current repository makes minor modifications to the OpenZepplin libraries such as combining multiple contracts from the library. From a development perspective, it is initially beneficial as it reduces the amount of effort. However, from a security perspective, it involves some risks as the cloned code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: We recommend using the library contracts directly unless there are compelling reasons to modify the contracts.

Update: [Re-audit 2] The auditors have recommended the team to address this issue, which was marked as "Unresolved" during the first re-audit. The team has responded as follows: "`OwnablePausable.sol` and `OwnablePausableUpgradeable.sol` are not actually cloned. `ERC20PermitUpgradeable.sol` and `ERC20Upgradeable.sol` are modified (some of the unused functions were removed from there)."

[Re-audit 1] From the team: "We will skip major code modifications for now and will keep that recommendation in mind in our future releases."

Automated Analyses

Slither

The Slither tool analyzed 53 contracts with 77 detectors, returning 187 results. These results were inspected manually and a large number of them were deteremined to be either false positive or insignificant. Three results remain, but are of information or best practices recommendation significance. The output for these three results are included below for reference.

```
Reentrancy in StakedEthToken._transfer(address,address,uint256) (contracts/tokens/StakedEthToken.sol#97-119):
        External calls:
        - (senderRewardsDisabled,recipientRewardsDisabled) = rewardEthToken.updateRewardCheckpoints(sender,recipient) (contracts/tokens/StakedEthToken.sol#103)
        State variables written after the call(s):
        - deposits[sender] = deposits[sender].sub(amount) (contracts/tokens/StakedEthToken.sol#115)
        - deposits[recipient] = deposits[recipient].add(amount) (contracts/tokens/StakedEthToken.sol#116)
        - distributorPrincipal = _distributorPrincipal (contracts/tokens/StakedEthToken.sol#112)
Reentrancy in StakedEthToken.mint(address,uint256) (contracts/tokens/StakedEthToken.sol#124-138):
        External calls:
        - rewardsDisabled = rewardEthToken.updateRewardCheckpoint(account) (contracts/tokens/StakedEthToken.sol#128)
        State variables written after the call(s):
        - deposits[account] = deposits[account].add(amount) (contracts/tokens/StakedEthToken.sol#135)
        - distributorPrincipal = distributorPrincipal.add(amount) (contracts/tokens/StakedEthToken.sol#131)
        - totalDeposits = totalDeposits.add(amount) (contracts/tokens/StakedEthToken.sol#134)
Reentrancy in StakedEthToken.toggleRewards(address,bool) (contracts/tokens/StakedEthToken.sol#79-92):
        External calls:
```

```
        - rewardEthToken.setRewardsDisabled(account,isDisabled) (contracts/tokens/StakedEthToken.sol#83)
        State variables written after the call(s):
        - distributorPrincipal = distributorPrincipal.add(accountBalance) (contracts/tokens/StakedEthToken.sol#88)
        - distributorPrincipal = distributorPrincipal.sub(accountBalance) (contracts/tokens/StakedEthToken.sol#90)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

Mythril

Mythril reported a total of 1 issue of High severity and 6 issues of Low severity. Of these, five Low severity issues are excluded through manual review due to insignificance. The remaining High and Low severity issues both involve the withdrawal function in the `./pool/PoolEscrow.sol` contract, which is currently a work-in-progress. The outputs are included below for reference.

```
==== External Call To User-Supplied Address ====
SWC ID: 107
Severity: Low
Contract: PoolEscrow
Function name: withdraw(address,uint256)
PC address: 1854
Estimated Gas Usage: 3797 - 39228
A call to a user-supplied address is executed.
An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an
intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.
--------------------
In file: @openzeppelin/contracts/utils/Address.sol:57

recipient.call{ value: amount }("")

--------------------
Initial State:

Account: [CREATOR], balance: 0x4c988026215932014, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}

Transaction Sequence: <excluded in this report>

==== Unprotected Ether Withdrawal ====
SWC ID: 105
Severity: High
Contract: PoolEscrow
Function name: withdraw(address,uint256)
PC address: 1854
Estimated Gas Usage: 3797 - 39228
Any sender can withdraw Ether from the contract account.
Arbitrary senders other than the contract creator can profitably extract Ether from the contract account. Verify the business logic carefully and make sure that appropriate security controls are in place to prevent
unexpected loss of funds.
--------------------
In file: @openzeppelin/contracts/utils/Address.sol:57

recipient.call{ value: amount }("")

--------------------
Initial State:

Account: [CREATOR], balance: 0x0, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x205ffffffff, nonce:0, storage:{}

Transaction Sequence: <excluded in this report>
```

Code Documentation

Update: The documentation has been improved substantially as of the second re-audit, with both points below addressed. The detailed specification for the permissioned version of the protocol can be found at https://docs.stakewise.io/smart-contracts-permissioned.

1. The documentation quality can be improved substantially as the current documentation does not clearly state use cases or functional requirements for some of the contracts. The current documentation is more user-facing and does not serve as a technical specification oriented towards developers. Thus, additional details would be needed to verify the intended behavior of the contracts to make an appropriate assessment of the smart contracts.

   In particular, the specification of the whitelisting functionality needs to be specified in greater detail. Information such as which actions can be performed by whitelisted users, which actions can be performed by non-whitelisted users, and edge cases such as what happens if a user moves from whitelisted to non-whitelisted, and vice-versa should be clearly answered.
2. The protocol relies on off-chain programs to a significant extent. While these components are indicated on the smart contracts architecture diagram of the public documentation, this information is still incomplete. We recommend clearly indicating the involvement of an off-chain program and its processes within the smart contract code comments for functions or contracts that rely on off-chain components (e.g., `Pool.stakeWithPartner()`, `Pool.stakeWithReferrer()`, the `Roles.sol` contract, or signature generation for the `Oracles.sol` contract).

Adherence to Best Practices

1. **[Fixed]** L12 of `./pool/Pool.sol` is a duplicate line.
2. **[Fixed]** `StakedEthToken.totalSupply()` can be declared external.
3. Contracts such as `./pool/Pool.sol` and `./Oracles.sol` use hard coded magic numbers in their logic (e.g., 1e4, 2, and 3). Instead of using hard coded numbers, we recommend assigning these values to appropriately named constants.

Test Results

Test Suite Results

The test suite has 207 passing tests and 6 pending tests. The test suite outputs from `yarn test` are reproduced below:

```
yarn run v1.22.15
$ hardhat test
└─> [DEBUG] Optimizer disabled. Unlimited contract sizes allowed.
└─> [DEBUG] Mainnet fork with block number 13952000
Compiling 53 files with 0.7.5
@openzeppelin/contracts/utils/Pausable.sol:32:5: Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient.
    constructor () internal {
    ^ (Relevant source part starts here and spans across multiple lines).

contracts/tokens/RewardEthToken.sol:21:1: Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet. Consider enabling the optimizer (with a low
"runs" value!), turning off revert strings, or using libraries.
contract RewardEthToken is IRewardEthToken, OwnablePausableUpgradeable, ERC20PermitUpgradeable {
^ (Relevant source part starts here and spans across multiple lines).

Compilation finished successfully


  Contract: Merkle Distributor
     ✓ not oracle fails to update merkle root (123ms)
    periodically distribute
       ✓ not admin fails to distribute tokens
       ✓ fails to distribute tokens with zero amount
       ✓ fails to distribute tokens from zero address
       ✓ fails to distribute tokens with max uint duration
       ✓ fails to distribute tokens with zero duration
       ✓ fails to distribute tokens without allowance
       ✓ fails to distribute when paused
       ✓ admin can distribute tokens (75ms)
    one time distribute
       ✓ not admin fails to distribute tokens
       ✓ fails to distribute tokens with zero amount
       ✓ fails to distribute tokens from zero address
       ✓ fails to distribute tokens without allowance
       ✓ fails to distribute when paused
       ✓ admin can distribute tokens (41ms)
    claim
       ✓ cannot claim when contract paused
       ✓ cannot claim when merkle root updating
       ✓ cannot claim with invalid merkle proof (70ms)
       ✓ cannot claim twice (240ms)
       ✓ cannot claim from not whitelisted address (243ms)
       ✓ can claim reward tokens (467ms)
      claiming within the same block
        - cannot claim after total rewards update in the same block
        - can claim before total rewards update in the same block
        - cannot claim before merkle root update in the same block
        - can claim after merkle root update in the same block

  Contract: Oracles
    assigning
       ✓ admin can assign oracle role to another account
       ✓ others cannot assign oracle role to an account
       ✓ oracles cannot assign oracle role to others
    removing
       ✓ anyone cannot remove oracles
       ✓ oracle cannot remove other oracles
       ✓ admins can remove oracles
    rewards voting
       ✓ fails to submit when contract is paused
       ✓ fails to submit with not enough signatures
       ✓ fails to submit with invalid signature
       ✓ fails to submit with repeated signature (70ms)
       ✓ fails to submit without oracle role assigned
       ✓ submits data with enough signatures (78ms)
    merkle root voting
       ✓ fails to submit when contract is paused
       ✓ fails to submit too early (61ms)
       ✓ fails to submit with not enough signatures
       ✓ fails to submit with invalid signature (79ms)
       ✓ fails to submit with repeated signature (48ms)
       ✓ fails to submit without oracle role assigned
       ✓ submits data with enough signatures
       ✓ fails to vote for total rewards and merkle root in same block (287ms)
    validator voting
       ✓ fails to submit when contract is paused
       ✓ fails to submit with not enough signatures (91ms)
       ✓ fails to submit with invalid signature (116ms)
       ✓ fails to submit with repeated signature (115ms)
       ✓ fails to submit without oracle role assigned
       ✓ can vote for multiple validators (143ms)

  Contract: PoolEscrow
     ✓ sets owner on the contract creation
     ✓ can receive ETH transfers
    commit ownership transfer
       ✓ owner can commit ownership transfer
       ✓ fails to commit ownership transfer if not an owner
       ✓ can commit ownership transfer to zero address
    apply ownership transfer
       ✓ future owner can apply ownership transfer
       ✓ fails to apply ownership transfer if not a future owner
       ✓ fails to apply ownership transfer if not committed
    withdraw ether
       ✓ owner can withdraw ether from the escrow
       ✓ fails to withdraw ether without admin role
       ✓ fails to withdraw ether with invalid payee address
       ✓ fails to withdraw ether when not enough balance

  Contract: Pool Validators
    add operator
       ✓ fails to add with not admin privilege (42ms)
       ✓ fails to add with zero operator address
       ✓ fails to add with invalid merkle root
       ✓ fails to add with invalid merkle proofs
       ✓ can update existing operator
       ✓ can add new operator (42ms)
    remove operator
       ✓ fails to remove by user other than admin and operator
       ✓ fails to remove not existing operator (49ms)
       ✓ operator or admin can remove operator
    commit operator
       ✓ fails to commit invalid operator
       ✓ fails to commit operator twice
       ✓ can commit operator
    register validators
       ✓ fails to register validator by not oracles
       ✓ fails to register validator for not committed operator (254ms)
       ✓ fails to register validator twice (223ms)
       ✓ fails to register for invalid operator (143ms)
       ✓ fails to register for invalid deposit data (150ms)
       ✓ fails to register with invalid validators deposit root (118ms)
       ✓ oracles can register one validator (160ms)
       ✓ oracles can register multiple validators (1135ms)

  Contract: Pool (settings)
    min activating deposit
       ✓ not admin fails to set min activating deposit
       ✓ admin can set min activating deposit
    pending validators limit
       ✓ not admin fails to set pending validators limit
       ✓ admin can set pending validators limit
       ✓ fails to set invalid pending validators limit
    activated validators
       ✓ not oracles contract or admin fails to set activated validators (49ms)
       ✓ admin can override activated validators
       ✓ oracles contract can set activated validators (189ms)

  Contract: Pool (stake)
     ✓ only PoolValidators contract can register new validators
     ✓ not admin cannot refund
     ✓ admin can refund
    stake
       ✓ fails to stake with zero amount
       ✓ fails to stake with zero address
       ✓ fails to stake in paused pool (101ms)
       ✓ fails to stake with not whitelisted address (96ms)
       ✓ mints tokens for users with deposit less than min activating (64ms)
       ✓ places deposit of user to the activation queue with exceeded pending validators limit (51ms)
      - activates deposit of user immediately with not exceeded pending validators limit
       ✓ can stake to different recipient address (38ms)
       ✓ can stake without recipient address
      staking with partner
         ✓ can stake with partner
         ✓ can stake with partner to different recipient address
      staking with referrer
         ✓ can stake with referrer
         ✓ can stake with referrer to different recipient address (42ms)
    activating
       ✓ fails to activate with invalid validator index (41ms)
       ✓ fails to activate in paused pool
       ✓ fails to activate not existing deposit (76ms)
       ✓ fails to activate deposit amount twice (46ms)
       ✓ activates deposit amount (198ms)
    activating multiple
       ✓ fails to activate with invalid validator indexes
       ✓ fails to activate in paused pool
       ✓ fails to activate not existing deposit (218ms)
       ✓ fails to activate multiple deposit amounts twice (58ms)
       ✓ activates multiple deposit amounts (66ms)

  Contract: OwnablePausable
    assigning admins
       ✓ admins can assign admin role to another account
       ✓ anyone cannot assign admin role to an account
    removing admins
       ✓ admin can remove himself
       ✓ admin can remove other admins
    assigning pausers
       ✓ admin can assign pauser role to another account
       ✓ others cannot assign pauser role to an account
       ✓ pausers cannot assign pauser role to others (101ms)
    removing pausers
       ✓ anyone cannot remove pausers
       ✓ pauser cannot remove other pausers
       ✓ admins can remove pausers (57ms)
    pausing
       ✓ pauser can pause contract
       ✓ pauser can unpause contract
       ✓ others cannot pause contract (39ms)
       ✓ others cannot unpause contract
  Contract: OwnablePausableUpgradeable
    assigning admins
       ✓ admins can assign admin role to another account
       ✓ anyone cannot assign admin role to an account
    removing admins
       ✓ admin can remove himself
       ✓ admin can remove other admins
    assigning pausers
       ✓ admin can assign pauser role to another account (39ms)
       ✓ others cannot assign pauser role to an account
       ✓ pausers cannot assign pauser role to others
    removing pausers
       ✓ anyone cannot remove pausers (58ms)
       ✓ pauser cannot remove other pausers
       ✓ admins can remove pausers
    pausing
       ✓ pauser can pause contract
       ✓ pauser can unpause contract
       ✓ others cannot pause contract
       ✓ others cannot unpause contract (90ms)

  Contract: Roles
    operators
       ✓ not admin fails to set operator
       ✓ fails to set operator to zero address
       ✓ fails to set operator with invalid revenue share (46ms)
       ✓ fails to set operator when paused
       ✓ admin can set operator
       ✓ fails to remove zero address operator
       ✓ fails to remove operator when paused (104ms)
    partners
       ✓ not admin fails to set partner
       ✓ fails to set partner to zero address
       ✓ fails to set partner with invalid revenue share
       ✓ fails to set partner when paused
       ✓ admin can set partner
       ✓ fails to remove zero address partner
       ✓ fails to remove partner when paused

  Contract: RewardEthToken
    restricted actions
       ✓ not admin fails to update protocol fee recipient address
       ✓ can set zero address for the protocol fee recipient
       ✓ admin can update protocol fee recipient address
       ✓ not admin fails to update protocol fee
       ✓ admin can update protocol fee
       ✓ fails to set invalid protocol fee
       ✓ only StakedEthToken contract can disable rewards (69ms)
    updateTotalRewards
       ✓ anyone cannot update rewards
       ✓ oracles can update rewards (140ms)
       ✓ anyone cannot update rewards
       ✓ oracles can update rewards (103ms)
       ✓ assigns protocol fee to distributor (107ms)
    transfer
       ✓ cannot transfer to zero address (87ms)
       ✓ cannot transfer to not whitelisted account (131ms)
       ✓ cannot transfer from zero address (67ms)
       ✓ can transfer zero amount (97ms)
       ✓ cannot transfer with paused contract (190ms)
       ✓ cannot transfer amount bigger than balance (125ms)
       ✓ can transfer rETH2 tokens to different account (84ms)
       ✓ cannot transfer rewards after total rewards update in the same block (329ms)
       ✓ can transfer rewards before total rewards update in the same block (161ms)

  Contract: StakedEthToken
    mint
       ✓ anyone cannot mint sETH2 tokens (66ms)
       ✓ updates distributor principal when deposited by account with disabled rewards (104ms)
    transfer
       ✓ cannot transfer to zero address (70ms)
       ✓ cannot transfer from zero address (92ms)
       ✓ cannot transfer to not whitelisted account (41ms)
       ✓ can transfer zero amount (41ms)
       ✓ cannot transfer with paused contract (45ms)
       ✓ cannot transfer amount bigger than balance (78ms)
       ✓ can transfer sETH2 tokens to different account
       ✓ preserves rewards during sETH2 transfer (218ms)
       ✓ updates distributor principal when transferring to account with disabled rewards (151ms)
       ✓ updates distributor principal when transferring from account with disabled rewards (60ms)
       ✓ does not update distributor principal when transferring between accounts with disabled rewards (243ms)
       ✓ cannot transfer staked amount after total rewards update in the same block (157ms)
       ✓ can transfer staked amount before total rewards update in the same block (148ms)

  Contract: StakedEthToken (toggle rewards)
    toggle rewards
       ✓ not admin cannot toggle rewards (59ms)
       ✓ fails to toggle rewards with the same value
       ✓ fails to toggle rewards with invalid account
       ✓ fails to toggle rewards with invalid account
       ✓ admin can toggle rewards (146ms)
       ✓ balance is not updated for disabled rewards account (651ms)
      - toggling rewards does not affect current rewards balance
    claim rewards
       ✓ not merkle distributor cannot claim rewards (40ms)

  Contract: WhiteListManager
    assigning managers
       ✓ admins can assign manager role to another account
       ✓ others cannot assign manager role to an account
       ✓ managers cannot assign manager role to others (70ms)
    removing managers
       ✓ anyone cannot remove managers (42ms)
       ✓ manager cannot remove other managers (81ms)
       ✓ admins can remove managers
    whitelist update
       ✓ cannot update whitelist when paused
       ✓ cannot update whitelist with invalid account (63ms)
       ✓ anyone cannot update whitelist
       ✓ manager can update whitelist


  207 passing (60m)
  6 pending

Done in 3705.24s.
```

Code Coverage

The commit as of the second re-audit (`b1e838f`) achieves 97.74% code coverage. The coverage results for this commit are available on CircleCI and the coverage report for this commit is available on Codecov.

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

```
7bd466bcd3803855bd20239345075b76c54e5e548a2a59a04c2e2963d694ae74 ./contracts/ContractChecker.sol
ca7fd86a903e39fc84b3cae9603fdaa3fcaa168964b15af1659409d09e95ecc0 ./contracts/Oracles.sol
9bda1f71c939abfa46f9778148f1c61113af306a3d3916b351fc6f024309f70c ./contracts/Roles.sol
31277c5faf2cf5a23a9306f6d83c47994421cfa9e93e5ad9e3c95a51c3981545 ./contracts/WhiteListManager.sol
9cf89c7d7ab9a3514d784ea33371a2c277537e27835677f180bcdde3409815e2 ./contracts/tokens/ERC20PermitUpgradeable.sol
425f1578d93e5d15d944693a2a4899230ff28d1d4a5c6f8806bbf69353b5159d ./contracts/tokens/ERC20Upgradeable.sol
0417e8dc2cb00a41ec3c8f775940c26fc06e53af7ee7e9101e19c51fefbe5da6 ./contracts/tokens/RewardEthToken.sol
692fa8049e7ee74adc16bf5d702553b35f4465c981bb6818ceba0dc9ae3f12ca ./contracts/tokens/StakedEthToken.sol
12e2cc65f533a000fac44c9e8dca7bca948195581f1f81de457cb93d2641a763 ./contracts/presets/OwnablePausable.sol
bc11095348fe9466914e5dca9d67ef137dd10c7554ced16466ccbdbb23b1a43a ./contracts/presets/OwnablePausableUpgradeable.sol
5b5545931ae032d84fcef8240f60629d8088f2ec3c819de84244e4caac9c4c9a ./contracts/pool/Pool.sol
0b73a0af85a00fda647cc1f5ccbe5211d5aa3da96c2e24a3a6852947b1c5112f ./contracts/pool/PoolEscrow.sol
31abfc9d5d3172a5528f216c475a7466e022950578ba18303304bfd0b6398263 ./contracts/pool/PoolValidators.sol
2ad6052edc04b410bfc0860abc84cd6f340152fac628f80e5907bd0e51208c86 ./contracts/mocks/ERC20Mock.sol
3975f4416ac6fa35ba00965cd1886b8c4e1d54a37cec9a50d764e2c937b4987b ./contracts/mocks/MulticallMock.sol
c880d92201d19dda4668409f2a74a08667dab027fe2747ea9d98a6ab72c689b6 ./contracts/mocks/OwnablePausableMock.sol
5c63bd04e250ed9ddd2fb4eb0406072c4ff175b202e354b6eff8181a6721077e ./contracts/mocks/OwnablePausableUpgradeableMock.sol
eca9707fc48e0c7854f21219ef0a2b79b82a0bbc2a1fb880f1afdb946c6f150a ./contracts/merkles/MerkleDistributor.sol
ab706f996c550608673bdd7c1fe82d1425939b4ca5437ea4b994df45a711ff2f ./contracts/interfaces/IDepositContract.sol
7ac2b1b6139dcf6f2ed95be402c856d527088dcc8c125e7c33034bd67ad66a36 ./contracts/interfaces/IMerkleDistributor.sol
4273a6b1cc41d41c6e2111547e6930a35396b3a3dbee6028525da8411a1bc4b6 ./contracts/interfaces/IOracles.sol
3f56ff9e1a1cd91b1ddc64f5bade8d98c6b27409949027e7c7ae488003cff705 ./contracts/interfaces/IOwnablePausable.sol
c397e2b01b1de18e21cff01d969c2376a6e47b38f775cf8e228a9c812653598f ./contracts/interfaces/IPool.sol
95371e32b765f2386e7e2d71245da09fea4f71310ecaf5e486e1c2aff2400a3a ./contracts/interfaces/IPoolEscrow.sol
16b00fd0fbd91921f711f31e27235b7ef181aca1337d7b7a70a67152ed00ff94 ./contracts/interfaces/IPoolValidators.sol
8f45207b65044bcee8bf18e4c3b4d57ad565d5daacfd61c7a5ac86bea4ace55a ./contracts/interfaces/IRewardEthToken.sol
b9467f03e8e86051f7985879519606ed2ce1ef50a35c88a2f72dd3e73c4f31d4 ./contracts/interfaces/IRoles.sol
c7dd617b29729a9fbc2e479d1ccd685e19a851d50c3208d4ed90a954001f7569 ./contracts/interfaces/IStakedEthToken.sol
f48972f3af46c11503923c5383c34ce99a1270315dc7edc4fc7e813e5621cb43 ./contracts/interfaces/IVestingEscrow.sol
a64b1d602465cefda2376379f054164c0ee9e864aff4b6e8ce9d2f89f4ef391c ./contracts/interfaces/IVestingEscrowFactory.sol
431fb0c7cdb05aa64711a2d926f5d616ce76880c1bbd8b1388cb7eca3e0bcd52 ./contracts/interfaces/IWhiteListManager.sol
```

Tests

```
7a65e41f6c9a6eb59022df71a9a835897a352e531c0a50f20d889f6efc9bdcb3 ./test/MerkleDistributor.test.js
3c22abec069bad996d60d2af2a673004161360f7819025172fc4aebb331dc703 ./test/Roles.test.js
0f4a399c7972d6607cc53e9321decb998d6d9eb0484329b075159e25e5aa25ac ./test/utils.js
af2af5ed1c76f561288a6513698bd844810763cd8160cb8f090c8a7c91c601ff ./test/WhiteListManager.test.js
efe5e78779b50809c44f44546fe535ef1658eb344e182f8dd7ce56b990ddfc19 ./test/tokens/RewardEthToken.test.js
62c017547b5a1043d9057bde174e9e6f3def9c169a7fa2f0ff2213b7bba866b5 ./test/tokens/StakedEthToken.test.js
14cd4b2a539a19e38969495b7c89f4e8c113e172b6b5a362620381d7ff23cee1 ./test/tokens/toggleRewards.test.js
27888bbadd500238d8b58d07c3573ddd71127468cd60dc97ffdb7a1bdc47f04f ./test/presets/OwnablePausable.test.js
34833350a39842e008ead563b4317fb668462a16bca242b42ecef1de8eb7006f ./test/presets/OwnablePausableTests.js
88777856c0d250b2e3f4c5d10605edc6236509a6263009c401561c25b548fe56 ./test/presets/OwnablePausableUpgradeable.test.js
63234011731a4b5295ee1000784c39b68bf0009360482896267cff9b30a0535c ./test/pool/depositDataMerkleRoot.js
0bf26cc1a409ffe2d21e7e98dd86c20265077ecb1aecc1beff9a03ee64bd6f3c ./test/pool/PoolEscrow.test.js
b55f9d286fa7db543479377b95363463813db7974064f92494835a3135e6af7e ./test/pool/PoolValidators.test.js
2cca6211e91585798c6715cf81af97337541822a4cedb12d86ccb1183e1271b3 ./test/pool/settings.test.js
606f9840ea2977f0888eb6c1346938272b284dd3e9a732acd5d583c5f73f309e ./test/pool/stake.test.js
b36441a1979019f2c97d8fe91273a0c5820287877ba326e0f7993a0eb8c25eed ./test/oracles/Oracles.test.js
```

Changelog

- 2022-04-22 - Initial report (commit `6ebf119`)
- 2022-05-04 - Re-audit report 1 (commit `5308be5`)
- 2022-05-06 - Re-audit report 2 (commit `b1e838f`)

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.