Quantstamp Security Assessment Certificate

May 17th 2022 — Quantstamp Verified

PlaySwoops

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

Executive Summary

Type	NFT Drop
Auditors	Mohsen Ahmadvand, Senior Research Engineer David Knott, Senior Research Engineer Fatemeh Heidari, Security Auditor
Timeline	2022-04-28 through 2022-04-28
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Medium
Test Quality	High

Source Code

Repository	Commit
swoops-nft	85c81f0
swoops-nft	a548213 (re-audit)
None	b01ce41 (re-re-audit)

Total Issues	12 (6 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	2 (0 Resolved)
Low Risk Issues	5 (4 Resolved)
Informational Risk Issues	2 (1 Resolved)
Undetermined Risk Issues	2 (0 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
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

In the course of the audit we identified 11 issues: one high, two medium, four low, two informational, and two undetermined severity issues. The identified re-entrancy vulnerability (high severity) can enable an attacker to circumvent the mint limits. Improper input validations on critical parameters makes it possible for the contract to be configured into a faulty state (medium severity). The owner is given the privilege to mint tokens with no restrictions (medium severity). Furthermore, the reviewed contracts do not follow the standard implementation of the proxy pattern.

ID	Description	Severity	Status
QSP-1	Re-entrancy in `SwoopsMint` Leads To Maximum Total Supply Violation	⇈ High	Fixed
QSP-2	Missing Input Validation	^ Medium	Acknowledged
QSP-3	Unrestricted Mint for Owners	^ Medium	Acknowledged
QSP-4	Unlocked Pragma	⌄ Low	Fixed
QSP-5	Surplus ETH Not Refunded	⌄ Low	Acknowledged
QSP-6	Missing Events	⌄ Low	Fixed
QSP-7	Missing Rescue Tokens	⌄ Low	Fixed
QSP-8	High Royalty Fees Can be Imposed Without Users' Consent	○ Informational	Acknowledged
QSP-9	`renounceOwnership` Can Lead to DoS	○ Informational	Fixed
QSP-10	Incompatible With Proxies	? Undetermined	Acknowledged
QSP-11	Vague Superuser Permissions	? Undetermined	Acknowledged
QSP-12	Use of `transferFrom` instead of `safeTransferFrom`	⌄ Low	Fixed

Quantstamp Audit Breakdown

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

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

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

Setup

Tool Setup:

- Slither v0.8.1

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Re-entrancy in `SwoopsMint` Leads To Maximum Total Supply Violation

Severity: *High Risk*

Status: Fixed

File(s) affected: `SwoopsMint.sol`, `SwoopsERC721.sol`

Description: A reentrancy exploit can occur when external contract calls are made. To protect against reentrancy, it is recommended to order one's functions by:
Checks - Perform validation checks. Effects - Update all contract state. Interactions - Make external contract calls. The above ordering protects one's functions from reentrancy because by the time external contract calls are made, and execution is given to a potentially untrusted contract, all contract state is already in the most up-to-date state.
`SwoopsMint` `mint` and `whitelistedMint` functions are vulnerable to reentrancy via `safeMint`'s `onERC721Received` check, which passes transaction execution to the address of the NFT minter. The NFT minter can then call `mint` or `whitelistedMint` again. This reentrancy exploit could result in Swoop NFT's `totalSupply` being greater than the `maxTokenSupply`.

Exploit Scenario: The exploit scenario below captures an attack on `whitelistedMint`.

1. PlaySwoops starting state:
    . `maxTokenSupply = 4`
    . `swoopsNftContract.totalSupply() = 2`
    . `maxTokensPerWhitelistedWallet = 3`
2. Attacker deploys a smart contract which calls `SwoopsMint`'s `whitelistedMint` function directly, and when `onERC721Received` is called on it.
3. Attacker contract calls `whitelistedMint` with a `quantity` of 2.
4. `SwoopsMint`'s validation checks pass as the current total supply of 2 and the mint quantity of 2 is less than or equal to the `maxTokenSupply` of 4.
5. `SwoopsMint` mints the attacker their first NFT and calls `onERC721Received`.
6. The attacker contract calls `whitelistedMint` with a `quantity` of 1.
7. `SwoopsMint`'s validation checks pass as the current `totalSupply` is 3 and the mint `quantity` being requested is 1. When they're added together, the result is 4, which is less than or equal to `maxTokenSupply`.
8. `SwoopMint` mints the attacker a second NFT, completing the nested mint call.
9. `SwoopMint` mints the attacker a third NFT, completing the initial mint call.

PlaySwoops ending state:

```
*  `maxTokenSupply = 4`
*  `swoopsNftContract.totalSupply() = 5`
*  `maxTokensPerWhitelistedWallet = 3`
```

Recommendation: Add OpenZeppelin's `ReentrancyGuard` to `mint` and `whitelistedMint`. Consider whether the gas savings from performing validations and then bulk minting are worth the check/effects/interactions pattern violation. If so, add technical documentation explaining that the decision to violate check/effects/interactions was intentional. If not, perform `canMint` checks before each NFT is minted.

QSP-2 Missing Input Validation

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `SwoopsMint.sol`, `SwoopsERC721.sol`

Description: Input validation for protocol configuration and administration is critical as the misseting of any parameters can cause the protocol not to behave as expected.
The following functions are missing input validation and can be misset:

- `SwoopsMint::L21(constructor)`
    . `nftContractAddress` doesn't check that the `nftContractAddress` is a contract and supports the `ERC721Royalty` interface.
- `SwoopsMint::L72(setMaxTokenSupply)`
    . Does not check that `newMaxTokenSupply` is greater than the current NFT token `totalSupply`.
- `SwoopsMint::L90(setPrice)`
    . Does not check that `newPrice` is not zero.
- `SwoopsMint::L97(setMerkleRoot)`
    . Does not check that `newRoot` is not an empty bytes string.
- `SwoopsMint::L104(setIsSaleActive)`
    . Does not check whether the sale is alreay `active`.
- `SwoopsMint::L111(setDirectMintEnabled)`
    . Does not check whether the direct mint is alreay `enabled`.
- `SwoopsERC721::L24(constructor)`
    . Does not check whether `baseURI` is an empty string.
- `SwoopsERC721::L35(setBaseUri)`
    . Does not check whether `uri` is an empty string.
- `SwoopsERC721::L63(setMintingContract)`

- . Does not check that `contractAddress` is a contract.

Recommendation: Add input validation checks to all the functions specified above.

Update: The Playswoops team has added input validation to the following functions:

- `SwoopsMint::L21(constructor)`
  . `nftContractAddress` is now checked to be a contract that implements the `IERC721` interface.
- `SwoopsMint::L72(setMaxTokenSupply)`
  . `newMaxTokenSupply` is now checked against `swoopsNftContract.totalSupply()` to ensure that it is never less than the current `swoopsNFT totalSupply`.
- `SwoopsERC721::L24(constructor)`
  . `baseURI` is now checked to have a length greater than zero.
- SwoopsERC721::L35(setBaseUri)
  . `uri` is now checked to have a length greater than zero.
- SwoopsERC721::L63(setMintingContract)
  . `contractAddress` is now checked to be a contract address.

The Playswoops team states that the lack of input validation on the following functions is by design:

- `SwoopsMint::L90(setPrice)`
- `SwoopsMint::L97(setMerkleRoot)`
- `SwoopsMint::L104(setIsSaleActive)`
- `SwoopsMint::L111(setDirectMintEnabled)`

QSP-3 Unrestricted Mint for Owners

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `SwoopsERC721.sol`

Description: The owner can mint an arbitrary number of tokens for themselves directly without going through the minting contract. The `safeMint` function has a `onlyOwnerOrMintingContract` modifier, which essentially enables the owner to bypass the minting contract and mint an arbitrary number of tokens at will. This way the owner can bypass the `maxTokenSupply` check that is part of the mining contract logic. That is, the check (`SwoopsMint.sol:138~141`) in the `canMint` modifier that limits mints per drop is completely circumvented:

```
        require(
             quantity + swoopsNftContract.totalSupply() <= maxTokenSupply,
             "Not enough tokens left to buy the requested quantity"
         );
```

Besides breaking the `maxTokenSupply` limit, this is certainly a privilege given to the development team and possibly unfavourable to the community.

Recommendation: Consider capping the number of tokens that the owner can mint without going through the minting contract. The owner minted tokens need to comply with the `maxTokenSupply` as well. Furthermore, ensure that this power (owner can mint as many tokens as they want) is explicitly reflected in the public-facing documentation.

Update: The Playswoops team states that "this is a known and expected part of the functionality. The initial iteration of this needs to be as flexible as possible, and that means having full control over how mints happen."

QSP-4 Unlocked Pragma

Severity: *Low Risk*

Status: Fixed

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".
All the solidity files reviewed specify a solidity pragma version of ^0.8.1 which means that they can be compiled with any minor solidity version release greater than or equal to 0.8.1.

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, i.e., change all solidity files' versions to `0.8.13` to lock the solidity version.

QSP-5 Surplus ETH Not Refunded

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `SwoopsMint.sol`

Description: As constructed, users may (accidentally) send extra ETH to mint functions. The contract does not return the surplus ETH to the caller.

Recommendation: Consider returning the surplus ETH to the caller in all mint functions.

Update: The Playswoop team states that "we are choosing to deal with overfunding manually if and when it happens."

QSP-6 Missing Events

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SwoopsMint.sol`

Description: Every operation should ideally emit an event (indexed properly). Complying with event emits makes logic implementations in the frontend significantly easier. The frontend can simply subscribe to an event of interest to inform the user about the status of their operations, e.g., the mint was successful and you minted the token XYZ.

Recommendation: Consider emitting events for major operations: mint, supply change, drop increase, status change (pause/unpause/sale/whitelistsale), etc.

Update: The Playswoop team added events to all major operations.

QSP-7 Missing Rescue Tokens

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SwoopsMint.sol`, `SwoopsERC721.sol`

Description: It is possible that some tokens accidentally get sent to contracts.

Recommendation: Consider introducing a method to sweep mistakenly sent tokens.

QSP-8 High Royalty Fees Can be Imposed Without Users' Consent

Severity: *Informational*

Status: Acknowledged

File(s) affected: `SwoopsERC721.sol:73`

Description: The owner can decide to impose high royalty fees at any time to all tokens in circulation. The owner can also set the receiver of the royalties globally.

Recommendation: Consider capping the royalty fees. The public facing documentation should explicitly state the fact that owners can change royalty fees at any time.

Update: The Playswoop team states that "this is a known and expected part of the functionality. The initial iteration of this needs to be as flexible as possible, and we aren't ready to fix a royalty right now- it's too early to know what the economics of the game demand."

QSP-9 `renounceOwnership` Can Lead to DoS

Severity: *Informational*

Status: Fixed

File(s) affected: `SwoopsMint.sol`

Description: As constructed the SwoopMint heavily relies on the owner role. The inherited Ownable contract comes with a `renounceOwnership` that sets address 0 as the owner. This may have catastrophic consequences on the project including funds being locked in the contract. A rogue insider can possibly trigger the function to harm the platform.

Recommendation: Consider overriding renounceOwnership.

QSP-10 Incompatible With Proxies

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `SwoopsERC721.sol`

Description: Constructors in Solidity can only be executed on contract deployment, therefore they are incompatible with proxies which can only run already deployed code.
The PlaySwoop's audit engagement document states that the "Relationship between the 721 and its proxy(ies)" is an area of concern. `SwoopsERC721`'s use of a `constructor`'s makes it unusable with proxies.

Recommendation: Modify `SwoopsERC721` to follow the initializer pattern by removing its `constructor` and changing all OpenZeppelin contract imports to be from OpenZeppelin's upgradeable repository.

Update: The Playswoops team states that they were aware that their contracts were not compatible with proxies and that this lack of compatibility was intentional.

QSP-11 Vague Superuser Permissions

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `SwoopsMint.sol`, `SwoopsERC721.sol`

Description: The `owner` role is used to configure, administer and collect protocol rewards for `SwoopsMint` and `SwoopsERC721` contracts. This lack of separation of responsibility creates a single point of failure. There's also no documentation on what safety controls will be in place for the `owner`.

Recommendation: Separate the configuration, administration, and collection of rewards from `SwoopsMint` and `SwoopsERC721` into distinct roles.

Update: The Playswoops team states that they are aware of the lack of granularity in their superuser permissions and that increasing the granularity is currently out of scope.

QSP-12 Use of `transferFrom` instead of `safeTransferFrom`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SwoopsMint.sol`

Description: `SwoopsMint`'s `withdrawToken` uses `transfer` to sweep ERC20 tokens from the `SwoopsMint` contract to the `owner`. However some non-standard ERC20's only return `false` instead of throwing an error when a transfer operation fails. Furthermore, some ERC20 tokens (e.g., USDT) do not return a boolean value at all. Since this is a sweeping function, no funds are at risk. Nevertheless, the emitted transfer events can be misleading.

Recommendation: Use OpenZeppelin's `SafeERC20` library's `safeTransfer` method when transferring ERC20 tokens.

Code Documentation

- [Fixed] `README.md` contains a `scaffold-eth` tutorial. Modify `README.md` only to contain information relevant to the PlaySwoops project to increase readability.
- [Fixed] While most of the code is well documented and quite straightforward, it is still good to document all code following the NatSpec Format for improved readability and maintainability.

Adherence to Best Practices

1. [Fixed] Some functions can be declared external to better optimize for gas

```
setBaseUri(string) should be declared external:
    - SwoopsERC721.setBaseUri(string) (contracts/SwoopsERC721.sol#35-37)
pause() should be declared external:
    - SwoopsERC721.pause() (contracts/SwoopsERC721.sol#41-43)
unpause() should be declared external:
    - SwoopsERC721.unpause() (contracts/SwoopsERC721.sol#47-49)
safeMint(address) should be declared external:
    - SwoopsERC721.safeMint(address) (contracts/SwoopsERC721.sol#54-58)
setMintingContract(address) should be declared external:
    - SwoopsERC721.setMintingContract(address) (contracts/SwoopsERC721.sol#63-65)
setRoyalty(address,uint96) should be declared external:
    - SwoopsERC721.setRoyalty(address,uint96) (contracts/SwoopsERC721.sol#73-78)
mint(uint256) should be declared external:
    - SwoopsMint.mint(uint256) (contracts/SwoopsMint.sol#28-34)
whitelistedMint(uint256,bytes32[]) should be declared external:
    - SwoopsMint.whitelistedMint(uint256,bytes32[]) (contracts/SwoopsMint.sol#42-69)
setMaxTokenSupply(uint256) should be declared external:
    - SwoopsMint.setMaxTokenSupply(uint256) (contracts/SwoopsMint.sol#74-76)
setMaxTokensPerWhitelistedWalletPerDrop(uint256) should be declared external:
    - SwoopsMint.setMaxTokensPerWhitelistedWalletPerDrop(uint256) (contracts/SwoopsMint.sol#81-85)
setPrice(uint256) should be declared external:
    - SwoopsMint.setPrice(uint256) (contracts/SwoopsMint.sol#90-92)
setMerkleRoot(bytes32) should be declared external:
    - SwoopsMint.setMerkleRoot(bytes32) (contracts/SwoopsMint.sol#97-99)
setIsSaleActive(bool) should be declared external:
    - SwoopsMint.setIsSaleActive(bool) (contracts/SwoopsMint.sol#104-106)
setDirectMintEnabled(bool) should be declared external:
    - SwoopsMint.setDirectMintEnabled(bool) (contracts/SwoopsMint.sol#111-113)
increaseWhitelistDropId() should be declared external:
    - SwoopsMint.increaseWhitelistDropId() (contracts/SwoopsMint.sol#117-119)
```

1. [Fixed] When transferring ownership, it is best to do that in two steps:
    - The present owner suggests a new address for the change.
    - In a separate transaction, the newly suggested address claims the ownership.

This two-step update enables for the correction of accidental proposals rather than leaving the system functioning with no/malicious owner.

Test Results

Test Suite Results

All tests are passing.

```
Contracts
  SwoopsERC721
    Initialization
       ✓ Constructor should not accept an empty baseurl (115ms)
    Minting
       ✓ Should be able to mint an NFT (98ms)
       ✓ Should be able to burn an NFT (118ms)
       ✓ Should return proper TokenURI (80ms)
    Non-owner interaction
       ✓ Non owners can't perform admin functions (212ms)
       ✓ Non-owners can perform public functions (41ms)
    Features paused
       ✓ Should disable ERC-721 transfer operations (197ms)
    Owner Invoking admin functions
       ✓ Should be able to pause and unpause (137ms)
       ✓ Should be able to set a baseURI (134ms)
       ✓ Can't set empty baseURI
       ✓ Can't set non-contract as minting contract
       ✓ Should be able to call renounceOwnership, which throws
       ✓ Should be able to nominate a new contract owner (73ms)
    ERC Standards
       ✓ Should support the NFT standard- ERC721
       ✓ Should support the royalty standard- ERC2981
       ✓ Should support the interface standard- ERC165
    Ownership transferral
       ✓ Cannot transfer ownership directly
       ✓ Cannot claim ownership if no one has been nominated
       ✓ Cannot claim ownership if someone else been nominated (45ms)
       ✓ Can claim ownership if you've been nominated
  SwoopsMint
    Initialization
       ✓ Constructor should only accept a 721-compliant contract
    Non-owner interaction
       ✓ Non owners can't perform admin functions (72ms)
    Other functions
       ✓ Owner should be able to get total treasury balance (53ms)
       ✓ Non-Owner should be able to get total treasury balance (54ms)
       ✓ The owner can withdraw funds from the treasury (61ms)
       ✓ Non-owner can not withdraw funds from the treasury (54ms)
    Admin functions
       ✓ Can withdraw other tokens (withdrawToken) (66ms)
       ✓ Should be able to set max token supply
       ✓ Mintable token supply must exceed total supply from the 721 (55ms)
       ✓ Should be able to set max tokens per drop
       ✓ Should be able to set price
       ✓ Should be able to set merkle root
       ✓ Should be able to set sale active
       ✓ Should be able to set direct mint enabled
       ✓ Should be able to increment the drop id
       ✓ Should be able to call renounceOwnership which throws
    Minting
      Direct Mint
         ✓ Can't mint when direct mint not enabled (44ms)
         ✓ Should be able to set royalty
         ✓ Can't mint when sales are not active
         ✓ Can't mint when incorrect tokens requested
         ✓ Can't mint when token supply too low (39ms)
```

```
          ✓ Can't mint when enough ether isn't sent (57ms)
          ✓ Mintable when when all criteria is met (78ms)
          ✓ Can't perform reentrancy attack to obtain extra NFT (105ms)
       Whitelist Mint
          ✓ Can't mint when sales are not active
          ✓ Can't mint when invalid number of tokens requested
          ✓ Can't mint when token supply too low
          ✓ Can't mint when enough ether isn't sent (58ms)
          ✓ Can't mint when sender not on the whitelist (45ms)
          ✓ Can't mint when sender sends incorrect whitelist proof (54ms)
          ✓ Can't mint when whitelist is empty (54ms)
          ✓ Mintable when criteria are met and proof is valid (62ms)
          ✓ Can mint on multiple drops (126ms)
          ✓ Can't perform reentrancy attack to obtain extra NFT (99ms)


  54 passing (11s)
```

Code Coverage

All the contracts in the audit scope come with a 100% test coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
OwnershipClaimable.sol	100	100	100	100	
SwoopsERC721.sol	100	100	100	100	
SwoopsMint.sol	**100**	**100**	**100**	**100**	

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

```
00d8d5772d3aa12c15dc2bfdcb3bd1507b60342f5d06c304a3d2ad10df6da23a ./contracts/SwoopsMint.sol

94a50cd894d91f03842de4575b69a0e5c26f0a9b5ea3bfe0bb46910c5e05f42e ./contracts/OwnershipClaimable.sol

7a75338e604967ec8c34c4dbd95e06438afa445df07628fb23cf403a27cc0292 ./contracts/SwoopsERC721.sol
```

Tests

```
19afe8c3c480a9f69f8f705e1ed19dda6a1d4288511accf990b09861dac1fb3a ./test/SwoopsERC721.js
```

Changelog

- 2022-04-29 - Initial report
- 2022-05-13 - Re-audit
- 2022-05-13 - Re-re-audit

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.