Quantstamp Security Assessment Certificate

August 15th 2022 — Quantstamp Verified

NFTMarket

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

Executive Summary

Type	NFT market place
Auditors	Kacper Bąk, Senior Research Engineer Ed Zulkoski, Senior Security Engineer
Timeline	2021-09-22 through 2022-04-05
EVM	London
Languages	Solidity
Methods	Architecture Review, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Undetermined

Source Code

Repository	Commit
None	None

Goals

- Can tokens get locked up in the contract?
- Can tokens be stolen from the contract?

Total Issues	**11** (4 Resolved)
High Risk Issues	**3** (2 Resolved)
Medium Risk Issues	**2** (0 Resolved)
Low Risk Issues	**4** (2 Resolved)
Informational Risk Issues	0 (0 Resolved)
Undetermined Risk Issues	**2** (0 Resolved)

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

During the review we uncovered a high number of issues spanning all severity levels. It is important to note that there is a relatively high number of high and medium severity issues. Furthermore, the project is incomplete (no framework to manage it), has no tests, features very poor documentation, and comes with no specification. Under no circumstanced do we recommend deploying the code as is.
Update: the team provided clarifications and addressed all of the issues. The test suite, however, is still missing and we highly recommend adding one.

ID	Description	Severity	Status
QSP-1	Missing Test Suite	⇈ High	Acknowledged
QSP-2	Reentrancy	⇈ High	Fixed
QSP-3	Auction Info May Be Overwritten for ERC1155 Tokens	⇈ High	Fixed
QSP-4	Use of External Tokens	^ Medium	Acknowledged
QSP-5	Dependence on external contracts through `fullSupportContract`	^ Medium	Acknowledged
QSP-6	Unlocked Pragma	⌄ Low	Fixed
QSP-7	Privileged Roles and Ownership	⌄ Low	Acknowledged
QSP-8	Dangerous Use of `_isContract()` Check	⌄ Low	Acknowledged
QSP-9	`getAddrDataFromContract()` and `getUintDataFromContract()` Silently Fail	⌄ Low	Fixed
QSP-10	Default Fees May Not Match Specification	? Undetermined	Acknowledged
QSP-11	Missing Values in `getAuctionInfo()`	? Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

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

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

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Toolset

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

Setup

Tool Setup:

- Slither v0.8.2

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 Missing Test Suite

Severity: *High Risk*

Status: Acknowledged

Description: The project has no test suite.

Recommendation: We highly recommend adding a test suite to ensure the code works as expected.

QSP-2 Reentrancy

Severity: *High Risk*

Status: Fixed

File(s) affected: `NFTMarket.sol`

Description: A reentrancy vulnerability is a scenario where an attacker can repeatedly call a function from itself, unexpectedly leading to potentially disastrous results. Here's a basic example representing the very attack which impacted The DAO in 2016:

```
    ```
    function withdraw_with_reentrancy(uint256 _amount) public {
        msg.sender.call.value(_amount)();   // ATTACKER CAN CALL AGAIN BEFORE
                                            // FUNCTION TERMINATES because `call`
                    // allows msg.sender to execute any code using fallback function
        balances[msg.sender] -= _amount;   // Balance only updated AFTER funds withdrawn
    }
    ```
```

Specifically, re-entrancy is present in `launchAuction()`, `launchReservedAuction()` due to the use of `transferToken()`.

Recommendation: Protect against re-entrancy attacks via the use of pattern Checks-Effects-Interactions and re-entrancy guards.

QSP-3 Auction Info May Be Overwritten for ERC1155 Tokens

Severity: *High Risk*

Status: Fixed

File(s) affected: `NFTMarket.sol`

Description: The `auctions` mapping is first indexed by the `contractAddr` followed by the `tokenId`. However, in the case of ERC1155 tokens, there may be more than 1 supply of each `tokenId`. This is apparent in the `token.safeTransferFrom(_from, _to, _tokenID, 1, DEFAULT_MESSAGE)` call, where the 1 indicates the amount of `_tokenID` tokens to transfer. Suppose a `(contractAddr, tokenId)` pair has a supply larger than 1, and an auction is launched for one instance of this token. If a new auction is started for a different instance, the `auctionInfo[contractAddr][tokenId]` data will be overwritten, wiping previous auction data, and eventually locking 1 instance of the token in the contract forever.

Recommendation: Ensure that auction data cannot be overwritten for tokens with multiple instances.

QSP-4 Use of External Tokens

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `NFTMarket.sol`

Description: External contracts are used throughout the project. If the external contracts do not behave as expected, an auction may be launched/finished/canceled without transferring the actual token (see lines 145 and 223, 441, 498, 525, 609, 687). Furthermore, the tokens may cause re-entrancy.

Recommendation: Inform users of potential issues and protect against re-entrancy attacks via the use of pattern Checks-Effects-Interactions and re-entrancy guards.

QSP-5 Dependence on external contracts through `fullSupportContract`

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `NFTMarket.sol`

Description: It is not clear which contracts will be registered as “full support contracts”, however the function `acceptOffer()` performs external calls to upon these contracts to invoke and `creatorProfitOf(uint256)`. If the contract is malicious, it may return profit values arbitrarily high.

Recommendation: Ensure that external contracts are trusted or inform users of potential risks.

QSP-6 Unlocked Pragma

Severity: *Low Risk*

Status: Fixed

File(s) affected: `NFTMarket.sol`

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, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-7 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `NFTMarket.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
Specifically, the owner may set future auction data, particularly the `storeDefaultProfit` value which may be arbitrarily high.

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

QSP-8 Dangerous Use of `_isContract()` Check

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `NFTMarket.sol`

Description: Several functions such as `launchAuction()` and `bidOrBuy()` have the check `!_isContract(msg.sender)`. However, `_isContract()` will incorrectly return `false` for contracts that are still under construction, allowing new contracts to bypass this check. The check should therefore not be relied upon as a security measure.

Recommendation: Ensure the contract adheres to best practices, such as the checks-effects-interactions pattern which mitigates reentrancy attacks, so that contracts cannot maliciously interact with each function.

QSP-9 `getAddrDataFromContract()` and `getUintDataFromContract()` Silently Fail

Severity: *Low Risk*

Status: Fixed

File(s) affected: `NFTMarket.sol`

Description: These two functions return 0 when the external call is not successful. This may lead to burned funds, particularly when `getAddrDataFromContract()` is used to transfer creator profit in the `acceptOffer()` function.
Further, it is not clear why the functions `getUintDataFromContractI()` and `getAddrDataFromContract()` exist at all. It seems the relevant functionality could be represented through an interface that expects the function signatures `creatorOf(uint256)` and `creatorProfitOf(uint256)`.

Recommendation: Declare an interface to support the required external function signatures, and check that the return values indicate success.

QSP-10 Default Fees May Not Match Specification

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `NFTMarket.sol`

Description: The specification states that the “contract also charges a handling fee of 3 percent at default”, and comments within the `AuctionInfo` struct on L62-63 suggest that 1% will go toward the creator and 2% will go to the store. However, the default `_creatorProfit` value is set to 0 on L154, and is only set to new value if `isFullSupportContract(_contractAddr)` is true. It is not clear based on the code provided if this correctly sets the creator fee to 1%.

Recommendation: Ensure that the creator fee is set appropriately.

QSP-11 Missing Values in `getAuctionInfo()`

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `NFTMarket.sol`

Description: The following `AuctionInfo` fields are missing from the return values:

1. `contractAddr`
2. `tokenID`
3. `createTime`

Is this due to callstack limits?
While `contractAddr` and `tokenId` can be obtained from `getAuctionContractInfo()`, `createTime` has no getter function and is not used anywhere.

Recommendation: Document why these fields were excluded. Remove `createTime` if unused.

Automated Analyses

Slither

Slither reported reentrancies in: `NFTMarket.acceptOffer()`, `NFTMarket.acceptOfferWithID()`, and `NFTMarket.bidOrBuy()`. We classified them as false positives.

Adherence to Specification

1. The code does not come with a proper specification.
2. It is unclear why `launchReservedAuction()` uses 1 for `_auctionEnd`. According to the spec the auction should end within 24h.
3. It is not clear why `bidOrBuy()` checks that `msg.value > auction.startPrice` in several cases as opposed to `>=`. The current approach would require natural prices such as 1 ETH to be exceeded by a single WEI.
4. In `transferToken()`, it is unclear what this snippet is needed for (used twice):

```
if (_to == address(this)) {
    require(
        token.isApprovedForAll(msg.sender, address(this)),
        "Approved not found"
    );
}
```

`safeTransferFrom()` is going to check approval regardless of if `_to == address(this)`.

Code Documentation

The code is poorly documented. Furthermore, there are a few typos:

1. In the functions `isDirectBuy()` and `isReservedAuction()`, `_auctionStratTime` should be `_auctionStartTime`.
2. `Creater` is used instead of the correct `creator` in several places.
3. On L473, `Caculate` should be `Calculate`.

Adherence to Best Practices

1. SafeMath not needed in Solidity 8.
2. The event `Log` is not used and likely leftover from testing.
3. Instead of importing `../node_modules/@openzeppelin`, a directory string such as `@openzeppelin/...` could be used instead.
4. The getter functions such as `getStoreDefaultProfit()` and `getExtraAuctionMinutes()` could be removed in favor of declaring the corresponding variable `public`.
5. Since `supportsInterface()` simply invokes `super.supportsInterface()`, it seems there is no need to implement it here.
6. In `launchAuction()` and `launchReservedAuction()`, the two checks `checkProfit(_creatorProfit)` and `checkProfit(_storeProfit)` are redundant, as it is later checked that `checkProfit(totalProfit)` holds, which sums both values.
7. In `acceptOffer()` on L621-622, the first `checkProfit(_creatorProfit)` is redundant, since the latter `checkProfit(_creatorProfit.add(storeDefaultProfit))` checks a value at least as big.
8. The function `setAuctionData()` should emit an event describing the configuration changes.
9. Since the variable `bytes private DEFAULT_MESSAGE` is not written anywhere, it could be declared as a constant.
10. The struct `offerData` should start with a capital letter like other structs.
11. Event fields should have descriptive names.
12. `uint` is used instead of `uint256` in some locations.

Test Results

Test Suite Results

No test suite was provided.

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

`d0e09728de2fcb8a3632e87278f04b804079603e692c259d26f490ef5db85e80 ./htc/NFTMarket.sol`

Changelog

- 2022-02-24 - Initial report
- 2022-04-05 - Revised 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.