Quantstamp Security Assessment Certificate

September 23rd 2022 — Quantstamp Verified

NFT Market (PresetHTC, Auction, Store)

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

Executive Summary

Type	NFT Market
Auditors	Roman Rohleder, Research Engineer Bohan Zhang, Auditing Engineer Fatemeh Heidari, Security Auditor
Timeline	2022-08-01 through 2022-08-05
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Low
Diff/Fork information	No repository was provided. The contract files have been provided by themselves. See the appendix for the file hashes that have been audited.

Source Code

Repository	Commit
None	None None

Total Issues	**17** (10 Resolved)
High Risk Issues	**2** (1 Resolved)
Medium Risk Issues	**4** (2 Resolved)
Low Risk Issues	**6** (4 Resolved)
Informational Risk Issues	**4** (3 Resolved)
Undetermined Risk Issues	**1** (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 audit we uncovered several issues covering all severity levels. The code came with little documentation, without a build environment and most importantly completely without tests. We strongly advise against deploying the project as it is in its current state.

Update: Following the fix verification, we determined that most of the issues have been fixed or sufficiently acknowledged. However, high severity issue QSP-1 ("Missing Test Suite") was only mitigated by providing some tests, which however cover less than 60% of the code base. We strongly recommend adding additional tests to improve coverage and ensure correct basic functionality in accordance with the developers planned design. The second high severity issue QSP-2 ("Copy and Sell / Instantly Buy with Low Price"), while explained to be by design, still holds a certain risk and we also recommend to consider. **Adherence to Specification:** All specification-related findings have been fixed, however the corresponding documentation is (at the time of finalization of this report) not yet publicly accessible. **Code Documentation:** All points, except one (5. Adding additional inline code comments) have been addressed. **Adherence to Best Practices:** About half of the recommendations have been implemented, further improving the code base, compared to its original state.

ID	Description	Severity	Status
QSP-1	Missing Test Suite	⇈ High	Mitigated
QSP-2	Copy and Sell / Instantly Buy with Low Price	⇈ High	Acknowledged
QSP-3	Potential Denial of Service for Rogue `share` Addresses	^ Medium	Fixed
QSP-4	Bidding on Cancelled Auctions Possible	^ Medium	Fixed
QSP-5	Multiple Auctions for Same Nft Leads to Lock of Funds	^ Medium	Acknowledged
QSP-6	Same Nft Tokenid in the Store/auction	^ Medium	Acknowledged
QSP-7	Dangerous Use of `_isContract()`	⌄ Low	Acknowledged
QSP-8	Missing Input Validation	⌄ Low	Mitigated
QSP-9	Privileged Roles and Ownership	⌄ Low	Mitigated
QSP-10	Uninitialized Upgradable Contract	⌄ Low	Fixed
QSP-11	Uncapped Fees	⌄ Low	Acknowledged
QSP-12	Seller's `LAUNCH_ROLE` Revocation Can Lead to Lock of Funds	⌄ Low	Fixed
QSP-13	Use of Unsafe Cast Potentially Leading to Truncation	○ Informational	Fixed
QSP-14	Application Monitoring Can Be Improved by Emitting More Events	○ Informational	Fixed
QSP-15	Clone-and-Own	○ Informational	Acknowledged
QSP-16	Timestamp Manipulation	○ Informational	Mitigated
QSP-17	Uninitialized Contract	? 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.

DISCLAIMER:
If the final commit hash provided by the client contains features that are not in scope of the audit or a re-audit, those features are excluded from the consideration in this report. In this regard, contract `MyProxy.sol` was added during the re-audit, which was out-of-scope for the audit and was provided, by the HTC team, as a mock contract for the added tests.

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. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Missing Test Suite

Severity: *High Risk*

Status: Mitigated

Description: To confirm basic functionality as well as all corner cases it is advised to have a compilable and passing test suite, ideally covering all lines and branches of the code at least once. The provided contracts came without build scripts and without tests, allowing for basic functionality to be broken unnoticed or introduce bugs.

Recommendation: We recommend using a development environment (i.e. Hardhat) for the given contracts and also add tests, covering all lines and branches at least once.

Update: Two test files have been provided (`testNFTAuction.js` and `testNFTStore.js`). However, the provided tests achieve only 58% statement coverage and 40% branch coverage. Both of which are way below the recommended minimum of 90%, leaving a lot of code completely untested, potentially hiding further functional/logical issues. We highly recommend adding more tests to achieve at least 90% coverage (and ideally more), before rolling the code out in production.

QSP-2 Copy and Sell / Instantly Buy with Low Price

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `NFTStore.sol`, `NFTAuction.sol`

Description: In contract `NFTStore.sol`, in function `launchProduct()`, LAUNCH_ROLE will add product into the system, and buyer can later pay for the product. The NFT contract will have to assign contract `NFTStore` as a mint role so that `NFTStore` can mint token for the buyer. However, such design can easily be exploited.
This also applies to `NFTAuction.launchAuction()`.

Exploit Scenario: In contract `NFTStore.sol`:
Seller `A` calls `launchProduct(_contractAddr=NFT_A, _Price = 100)` and launches his product. Seller `B` saw this, and calls `launchProduct(_contractAddr=NFT_A, _Price = 99)`, and launches exact same product as A's. But since the price is lower, buyer will buy `B`'s product. And since `NFT_A` allows `NFTStore` to mint, the buying operation will succeed. Even worse, `B` could launch it with price = 1 in vip mode, and buy it himself.
As in contract `NFTAuction`, seller can copy an auction from another seller with a shorter end time to call `finalizeAuction()` earilier than the victim auction. If he wants this NFT, he could launch auction in vip mode, place a bid, finalize the auction in one hour.

Recommendation: Reconsider the protocol design by focusing on NFT contract and minter role. One valid way is to add a mapping structure inside `ERC721` contract to keep the record of addresses who are valid seller. And this record could be used when launching the product/auction.

Update: Acknowledged with: `Actually, the NFTStore/NFTAuction are designed for only 1 seller (HTC only). We have updated this in the document that the 2 contracts shouldn't be shared between different sellers. Therefore we also removed seller verification in the source code.`
However, there is still potential risk since `Default Admin` can grant/revoke `LAUNCH_ROLE`. If they designed that there is only 1 `LAUNCH_ROLE`. A solid way would be to override `AccessControl.sol`, and make sure `LAUNCH_ROLE` cannot be revoked, and cannot be granted if there is already one.

QSP-3 Potential Denial of Service for Rogue `share` Addresses

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `NFTStore.sol`

Description: In contract `NFTStore.sol` the native `.transfer()` function is used to transfer ETH, which only forwards `2300` gas and reverts on error. As during `vipBuy()` and `Buy()` funds are always forwarded to all shares, a malicious `shareAddr` could prevent the sell operation, by consuming more than `2300` gas on ETH receival or otherwise err. The same applies for the `storeAddress` and `product.seller` address.

Recommendation: Consider a similar withdraw-on-error system as already implemented in `NFTAuction.sol`.

Update: Fixed, by implementing a withdraw-on-error system for said functions, as suggested.

QSP-4 Bidding on Cancelled Auctions Possible

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `NFTAuction.sol`

Description: Functions `NFTAuction.vipBid()` and `NFTAuction.Bid()` only check if the bid is within the allowed time window of the auction, but does not take into account the status of the auction (Unsold), allowing one to bid on cancelled auctions so long as it falls in the allowed time frame.

Recommendation: Add a check on the auction status inside `vipBid()` and `Bid()` to make sure it is `Open`.

Update: Fixed, by adding a check on the auction to be in `Open` state, as suggested.

QSP-5 Multiple Auctions for Same Nft Leads to Lock of Funds

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `NFTAuction.sol`

Description: There is no control that a unique token(combination of _contractAddr and _tokenId) is not launched in multiple auctions. If same token is listed in different actions, bidders send their funds to the contract on different auctions of the token. But the finalizeAuction will be executed successfully for just one of the auctions, and others will fail due to trying to mint an existing token and the funds get locked in the contract. Similar scenario leads to lock of the funds if a previousely minted token is launcehd in an auction.

Recommendation: Add appropriate controls to make sure a token is not listed in different auction. In addition add controls to make sure minted tokens will not be listed in auctions.

Update: Acknowledged with: `HTC is the only seller and takes control of the token IDs through our back-end server. We decide not to handle this in smart contract.`

QSP-6 Same Nft Tokenid in the Store/auction

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `NFTStore.sol`, `NFTAuction.sol`

Description: In `NFTStore.sol`, Two product `A`, `B` can have the same contract address and `productTokenIds[A]`, `productTokenIds[B]` may have the same TokenID. So, when that token is minted in the product `A`. The token cannot be minted again in `B`, which means that product exists but is invalid to be bought. Although this would not harm the security, since the transaction will revert in the end. But this gives users bad using experience.
Similarly in `NFTAuction.sol`, two auctions for the same NFT can exist at the same time. The situation in `NFTAuction` is even worse since this will result a waste of gas from multiple parties. And the seller has to cancel an auction to return ether to the bidder if the token can no longer be minted. If the seller refuse to cancel the auction, the fund used for bid will be **locked** in the contract.

Recommendation: Consider keeping a record of token ID in the `NFTStore`, `NFTAuction` contract.

Update: Acknowledged with: `HTC is the only seller and takes control of the token IDs through our back-end server. We decide not to handle this in smart contract.`

QSP-7 Dangerous Use of `_isContract()`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `NFTStore.sol`

Description: Functions `vipBuy()` and `Buy()` have the check `require(!_isContract(msg.sender), "Illegal operation from contract.");`. However, `_isContract()` will incorrectly return `false` for contracts calling from within the constructor, allowing them 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 mitigate reentrancy attacks, so that contracts cannot maliciously interact with each function.

Update: Acknowledged with: `We know that isContract() is not secure enough. We also implement check-effect-interaction and use nonReentrant modifier as well.`

QSP-8 Missing Input Validation

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `NFTAuction.sol`, `NFTStore.sol`

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following functions do not have a proper validation of input parameters:

1. `NFTAuction.__NFTAuction_init()` does not check that parameter `_storeAddress` is different from `address(0)`.
2. `NFTAuction.__NFTAuction_init()` does not check that parameter `_storeProfit` is smaller than `BASE_PERCENT` (i.e. by calling `checkProfit()` on it). (**Update:** Fixed)
3. `NFTAuction.launchAuction()` does not check that parameter `_contractAddr` is different from `address(0)`.
4. `NFTAuction.launchAuction()` does not check that parameter `tokenID` has not yet been minted.
5. `NFTAuction.launchAuction()` does not check that parameter `_shareProfit` contains different and non-zero addresses.
6. `NFTAuction.batchLaunchAuctions()` does not check that parameter `_contractAddr` is different from `address(0)`.
7. `NFTAuction.batchLaunchAuctions()` does not check that parameter `tokenIds` have not yet been minted.
8. `NFTAuction.batchLaunchAuctions()` does not check that parameter `_shareProfit` contains different and non-zero addresses.
9. `NFTAuction.setAuctionData()` does not check that parameter `_storeAddress` is different from `address(0)`.
10. `NFTAuction.setAuctionLocked()` does not check that the given auction has not already been finalized or cancelled. (**Update:** Fixed)
11. `ERC721PresetHTC.grantMinterRole()` does not check that parameter `account` is different from `address(0)`.
12. `NFTStore.__NFTMarket_init()` does not check that parameter `_storeAddress` is different from `address(0)`.
13. `NFTStore.__NFTAuction_init()` does not check that parameter `_storeProfit` is smaller than `BASE_PERCENT` (i.e. by calling `checkProfit()` on it). (**Update:** Fixed)
14. `NFTStore.launchProduct()` does not check that parameter `_contractAddr` is different from `address(0)`.
15. `NFTStore.launchProduct()` does not check that parameter `_Price` is non-zero. (**Update:** Fixed)
16. `NFTStore.launchProduct()` does not check that parameter `_totalSupply` is non-zero. (**Update:** Fixed)
17. `NFTStore.launchProduct()` does not check that parameter `_shareProfit` contains different and non-zero addresses.
18. `NFTStore.launchProduct()` does not check that parameter `_productTokenIds` contains different token IDs, which are also not yet minted.
19. `NFTStore.setProductLocked()` does not check that the product has not already been sold out or cancelled. (**Update:** Fixed)
20. `NFTStore.setStoreData()` does not check that parameter `_storeAddress` is different from `address(0)`.
21. `merkleProof` in `vipBid()` function and `_merkleRoot` in `setMerkleRoot()` function of both `NFTAuction.sol` and `NFTStore.sol` contracts should not be empty `bytes32`.

Recommendation: Consider adding according input checks.

QSP-9 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `NFTAuction.sol`, `NFTStore.sol`, `ERC721PresetHTC.sol`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.
The `NFTAuction.sol` contract contains the following privileged roles:

- `DEFAULT_ADMIN_ROLE`, as initialized during the `__NFTAuction_init()` call:
  - Renounce the role and thereby disable all followingly listed actions, by calling `renounceRole()`.
  - Add/remove addresses from arbitrary roles and define role admins, by calling `grantRole()`, `revokeRole()` and `_setRoleAdmin()`.
- `CONFIGURATOR_ROLE`, as initialized during the `__NFTAuction_init()` call:
  - Renounce the role and thereby disable all followingly listed actions, by calling `renounceRole()`.
  - Modify critical contract variables (`storeAddress`, `storeDefaultProfit`, `extraAuctionMinutes`, `priceThreshold` and `priceLimit`), by calling `setAuctionData()` and `setPriceLimit()`.
- `LAUNCH_ROLE`, as initialized during the `__NFTAuction_init()` call:

-
  - Renounce the role and thereby disable all followingly listed actions, by calling `renounceRole()`.
  - Launch auctions, by calling `launchAuction()` or `batchLaunchAuctions()`.
  - Finalize/Terminate finished auctions, by calling `finalizeAuction()`.
  - Cancle it's own listed auctions, by calling `cancelAuction()`.
  - Define 'VIP' members for its listed auctions, by calling `setMerkleRoot()`.
  - Lock/Unlock it's own auctions for 'VIPs' only, by calling `setAuctionLocked()`.

The `NFTStore.sol` contract contains the following privileged roles:

- `DEFAULT_ADMIN_ROLE`, as initialized during the `__NFTMarket_init()` call:
  - Renounce the role and thereby disable all followingly listed actions, by calling `renounceRole()`.
  - Add/remove addresses from arbitrary roles and define role admins, by calling `grantRole()`, `revokeRole()` and `_setRoleAdmin()`.
- `CONFIGURATOR_ROLE`, as initialized during the `__NFTMarket_init()` call:
  - Renounce the role and thereby disable all followingly listed actions, by calling `renounceRole()`.
  - Modify critical contract variables (`storeAddress` and `storeDefaultProfit`), by calling `setStoreData()`.
- `LAUNCH_ROLE`, as initialized during the `__NFTMarket_init()` call:
  - Renounce the role and thereby disable all followingly listed actions, by calling `renounceRole()`.
  - Launch NFT sells, by calling `launchProduct()`.
  - Cancle it's own listings, by calling `cancelProduct()`.
  - Define 'VIP' members for its listed auctions, by calling `setMerkleRoot()`.
  - Lock/Unlock it's own auctions for 'VIPs' only, by calling `setProductLocked()`.

The `ERC721PresetHTC.sol` contract contains the following privileged roles:

- `owner`, as initialized during the `constructor()` call:
  - Renounce the role and thereby disable all followingly listed actions, by calling `renounceOwnership()`.
  - Transfer the role to an arbitrary other address, by calling `transferOwnership()`.
  - Modify the contract URI (`contractUri`), by calling `setContractUri()`.
  - Modify the base URI (`_baseTokenURI`), by calling `setBaseUri()`.
  - Modify token URIs (`_tokenURIs[]`), by calling `setTokenURI()`.
  - Add/Remove arbitrary addressess as minters, by calling `grantMinterRole()`/`revokeMinterRole()`.
- `minters[]`, as set via `grantMinterRole()` and checked via `hasMinterRole()`:
  - Mint new tokens to arbitrary addresses, by calling `mint()`.

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

Update: The mentioned roles and privileges have been documented in privately shared documentation (`NFT Market 2.0.pdf`). At the time of finalization of this report this documentation was however not yet publicly accessible.

QSP-10 Uninitialized Upgradable Contract

Severity: *Low Risk*

Status: Fixed

File(s) affected: `NFTStore.sol`, `NFTAuction.sol`

Description: Avoid leaving a contract uninitialized. An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation contract, which may impact the proxy.

Exploit Scenario: The attacker could front run the initializing transaction and take control of the contract.

Recommendation: To prevent the implementation contract from being used, you should invoke the `_disableInitializers()` function in the constructor to automatically lock it when it is deployed.

Update: Fixed, by adding a constructor with the `initializer` modifier.

QSP-11 Uncapped Fees

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `NFTStore.sol`, `NFTAuction.sol`

Description: Functions `NFTAuction.setAuctionData()` and `NFTStore.setStoreData()` are having no further constraints on the `_storeProfit` other than 100%. Consequently, users can be subject to very high fees. This can subject users to high fees with no prior announcments.

Recommendation: Consider adding a hardcoded 'sane' upper bound for fees, enforce this bound in said functions and communicate this bound to users in public facing documentation.

Update: Acknowledged with: `HTC is the only seller and also the owner of platform. We decide not to set upper bound of the fee by requirement. Add this information in the document.`

QSP-12 Seller's `LAUNCH_ROLE` Revocation Can Lead to Lock of Funds

Severity: *Low Risk*

Status: Fixed

File(s) affected: `NFTAuction.sol`

Description: In the `finalizeAuction()` function, the caller is the `seller` of the auction and also has the `LAUNCH_ROLE`. The seller should not have access to launch new auctions, but should still be able to finalize their previous auctions. If before calling `finalizeAuction()` the admin revokes `LAUNCH_ROLE` for the `seller` the auction remains in open status and funds get locked in the contract. This is the result of mixing seller and launcher roles in one `LAUNCH_ROLE` role.

Recommendation: Revise the roles and the access control of functions to prevent this situation. One mitigation might be a more granular role definition to separate the launcher role from the seller role.

Update: Fixed, as per following acknowledgement and its entailing changes: `According to QSP-2, Contract is designed for only 1 seller. We removed seller verification and let all accounts with LAUNCH_ROLE can manage auctions.`

QSP-13 Use of Unsafe Cast Potentially Leading to Truncation

Severity: *Informational*

Status: Fixed

File(s) affected: `NFTStore.sol`

Description: Storage variable `storeDefaultProfit` in contract `NFTStore.sol` is stored as an `uint256` variable, although, by business, it should not be able to exceed `10**4` (`BASE_PERCENT`). In L98 a primitive cast operation is performed `uint16(storeDefaultProfit)`, potentially leading to truncation, as the aforementioned bound is not enforced during `__NFTMarket_init()` and `storeDefaultProfit` could exceed `type(uint16).max`.

Recommendation: We recommend considering storing `storeDefaultProfit` as `uint16` and/or enforcing `storeDefaultProfit` to be within bounds inside `__NFTMarket_init()`.

Update: Fixed, by enforcing `storeDefaultProfit` is within bounds inside `__NFTMarket_init()`, as suggested.

QSP-14 Application Monitoring Can Be Improved by Emitting More Events

Severity: *Informational*

Status: Fixed

File(s) affected: `ERC721PresetHTC.sol`, `NFTAuction.sol`, `NFTStore.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transition can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs, or hacks. Below we present a non-exhaustive list of events that could be emitted to improve the application management:

1. `ERC721PresetHTC.setContractUri()` does not emit an event reflecting changes made to the state variable `contractUri`. (**Update:** Fixed)
2. `ERC721PresetHTC.setBaseUri()` does not emit an event reflecting changes made to the state variable `_baseTokenURI`. (**Update:** Fixed)
3. `ERC721PresetHTC._setTokenURI()` does not emit an event reflecting changes made to the state variable `_tokenURIs[]`. (**Update:** Fixed)
4. `NFTAuction.setPriceLimit()` does not emit an event reflecting changes made to the state variables `priceThreshold` and `priceLimit`. (**Update:** Fixed)
5. `NFTAuction.setMerkleRoot()` does not emit an event reflecting changes made to the state variable `merkleRoots[]`. (**Update:** Fixed)
6. `NFTStore.setStoreData()` does not emit an event reflecting changes made to the state variables `storeAddress` and `storeDefaultProfit`. (**Update:** Fixed)
7. `NFTStore.setMerkleRoot()` does not emit an event reflecting changes made to the state variable `merkleRoots[]`. (**Update:** Fixed)

Recommendation: Consider emitting the events.

QSP-15 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `NFTAuction.sol`, `NFTStore.sol`

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

In particular, OpenZeppelins `ReentrancyGuard` code as well as `Pausable` code is cloned-and-owned inside `NFTAuction.sol` and `NFTStore.sol`.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve traceability of the file.

Update: Acknowledged with: `We decide not to change design in this stage.`

QSP-16 Timestamp Manipulation

Severity: *Informational*

Status: Mitigated

File(s) affected: `NFTAuction.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps up to around 15 seconds for their own purposes. If a smart contract relies on a timestamp, it must be taken into account.

Recommendation: Dev team can clarify to users that 15 seconds difference would not affect the functionality and security of the protocol.

Update: The mentioned potential time manipulation has been documented in privately shared documentation (`NFT Market 2.0.pdf`). At the time of finalization of this report this documentation was however not yet publicly accessible.

QSP-17 Uninitialized Contract

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `NFTStore.sol`, `NFTAuction.sol`

Description: `NFTAuction.sol` and `NFTStore` contracts inherit from `AccessControlUpgradeable` contract which is never initialized. Although the `__AccessControl_init` has no implementation but the contract is upgradable and might change in future.

Recommendation: We recommend initializing `AccessControlUpgradeable` by calling `__AccessControl_init` in the `__NFTMarket_init(..)` and `__NFTAuction_init` functions.

Update: Acknowledged with: `Since __AccessControl_init has no implementation, we decide not to modify at this stage.`

Automated Analyses

Slither

We were unable to run slither on the given files, due to the non-standard project environment.

Adherence to Specification

1. The specification for `NFTAuction.sol` states that `Contract also charges a platform handling fee of 3 percent at default`. However, this is not reflected in the code. (**Update:** Fixed, but documentation not yet publicly accessible)
2. Following inconsistencies between the documentation and the code has been noted:
    1. `cancelAuction (uint256 _auctionID)` is in the document, but in the code base, it should be `cancelProduct(uint256 _productID)` (**Update:** Fixed, but documentation not yet publicly accessible)
    2. `finalizeAuction (uint256 _auctionID)` is missing in the code base. (**Update:** Fixed, but documentation not yet publicly accessible)
    3. `WithdrawRefund()` is missing in the code base. (**Update:** Fixed, but documentation not yet publicly accessible)

Code Documentation

1. Misleading code comment on L34 of `ERC721PresetHTC.sol`, stating `nextTokenId is initialized to 1, since starting at 0 leads to higher gas cost for the first minter`. However, the following code does only initialize `contractUri`. (**Update:** Fixed)
2. The `require()` error message on L101 of `ERC721PresetHTC.sol` states `ERC721: batchBurn caller is not owner`. However, the corresponding function is the normal `burn()` and not `batchBurn()` function. (**Update:** Fixed)
3. Missing or incorrect NatSpec comments:
    1. `NFTAuction.getAuctionInfo()`: Missing NatSpec comments for return values. (**Update:** Fixed)
    2. `NFTAuction.getShareProfitInfo()`: Missing NatSpec comments for return value. (**Update:** Fixed)
    3. `NFTAuction.vipBid()`: Missing NatSpec comments for return value. (**Update:** Fixed)
    4. `NFTAuction.Bid()`: Missing NatSpec comments for return value. (**Update:** Fixed)
    5. `NFTStore.launchProduct()`: Missing NatSpec comments for return value. (**Update:** Fixed)
    6. `NFTStore.vipBuy()`: Missing NatSpec comments for return value. (**Update:** Fixed)
4. The following typographical errors have been noted:
    1. L284 of `NFTAuction.sol`: `for indentify` -> `to verify the`. (**Update:** Fixed)
    2. L294 of `NFTAuction.sol`: `start` -> `started`. (**Update:** Fixed)
    3. L302 of `NFTAuction.sol`: `didn't` -> `isn't`. (**Update:** Fixed)
    4. L311 of `NFTAuction.sol`: `didn't` -> `isn't`. (**Update:** Fixed)
    5. L350 of `NFTAuction.sol`: `start` -> `started`. (**Update:** Fixed)
    6. L358 of `NFTAuction.sol`: `didn't` -> `isn't`. (**Update:** Fixed)
    7. L367 of `NFTAuction.sol`: `didn't` -> `isn't`. (**Update:** Fixed)
    8. L407 of `NFTAuction.sol`: `end` -> `ended`. (**Update:** Fixed)
    9. L536 of `NFTAuction.sol`: `limitaion` -> `limitation`. (**Update:** Fixed)
    10. L113 of `NFTStore.sol`: `< 100%` -> `<= 100%`. (**Update:** Fixed)
5. Consider adding documentation to the code base directly. Especially for the parameter explanation and function purpose explanation.

Adherence to Best Practices

1. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
    1. `ERC721PresetHTC.SpecialBurn()`,
    2. `ERC721PresetHTC.MinterRoleChanged()`,
    3. `NFTAuction.AuctionLaunched()`,
    4. `NFTAuction.NewBidReceived()`,

5. `NFTAuction.AuctionEnded()`,
6. `NFTAuction.AuctionCancelled()`,
7. `NFTAuction.Paused()`,
8. `NFTAuction.Unpaused()`,
9. `NFTAuction.AuctionDataChanged()`,
10. `NFTAuction.LogSendFail()`,
11. `NFTAuction.LogWithdrawRefund()`,
12. `NFTAuction.SetAuctionLocked()`,
13. `NFTStore.ProductLaunched()`,
14. `NFTStore.ProductCancelled()`,
15. `NFTStore.ProductSold()`,
16. `NFTStore.Paused()`,
17. `NFTStore.Unpaused()`,
18. `NFTStore.SetProductLocked()`.

2. To ensure interface compatibility, contracts should inherit from their interfaces contracts. Therefore the following changes should be considered:
    1. Contract `ERC721PresetHTC.sol` should inhert from `IERC721HTC.sol` (and `IERC721HTC.sol` should not inherit from `IERC721Upgradeable`, as it is unused).

3. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but instead declare them once (as constant and commented) and use these constant variables instead. Following instances should therefore be changed accordingly:
    1. L133 of `NFTAuction.sol`: `31536000`. (**Update:** Fixed)
    2. L134 of `NFTAuction.sol`: `3600` (consider using the keyword `hours`). (**Update:** Fixed)
    3. L135 of `NFTAuction.sol`: `31536000` (consider using the keyword `days`). (**Update:** Fixed)

4. For improved readability and code quality it is advised to remove duplicate or unused code. In this regard consider the following cases:
    1. Contracts `NFTAuction.sol` and `NFTStore.sol` import and used `SafeMathUpgradeable.sol`. However, since both contracts enforce solidity version `0.8.0`, which implicitly performs the same arithmetic bounds checks, the use of `SafeMathUpgradeable.sol` becomes redundant.
    2. Code in L567-L590 of `NFTAuction.sol` (code related to `pause`-functionality) is unused and should therefore be removed.
    3. `NFTStore.getNow()` is unused and should therefore be removed. (**Update:** Fixed)

5. To prevent errors during further development it is advised to not have large blocks of duplicate code and instead re-use common code. In this regard, consider the refactoring the following instances:
    1. Functions `NFTAuction.vipBid()` and `NFTAuction.Bid()` share the same code, with the exception of the modifier `isValidMerkleProof()` and `auctionLocked[_auctionID]` check.
    2. Functions `NFTStore.vipBuy()` and `NFTStore.Buy()` share the same code, with the exception of the modifier `isValidMerkleProof()` and `productLocked[_productID]` check.
    3. Contracts `NFTAuction.sol` and `NFTStore.sol` share many identical variables (`CONFIGURATOR_ROLE`, `LAUNCH_ROLE`, `BASE_PERCENT`, `storeAddress`, `storeDefaultProfit`, `shareProfits`), the struct `ShareProfit` and functions `isValidMerkleProof()` and `checkProfit()`. Consider therefore creating a common abstract class, of which both contracts inherit.

6. Before rolling out code in production, any pending `TODO` items in code should be resolved in order to not deploy potentially unfinished code. In this regard the following `TODO` items still remain in code and should be resolved:
    1. L194 of `NFTStore.sol`: `TODO:rename startPrice` (**Update:** Fixed)

7. `BASE_PERCENT` can be defined as constant.
8. In functions `NFTStore.Buy()`,`NFTStore.vipBuy()` and `NFTAuction.finalizeAuction()`: If L196, L251 and L425 respectively are changed to to `uint256 amountForShareProfit = amount;`, some gas can be saved.
9. When declaring the variable as `memory`, you'll have to pay extra gas for reserving the memory. In the code, there is some variables are declared as `memory`, which is not nessesary. Switching `memory` to `storage` would help save gas.
    1. `NFTStore.getProductInfo()` (**Update:** Fixed)
    2. `NFTAuction.getAuctionContractInfo()` (**Update:** Fixed)
    3. `NFTAuction.getAuctionInfo()` (**Update:** Fixed)
       Although, they are `view` functions. But these also might be used in other contract's functions.

10. In function `NFTStore.setProductLocked()`, it is not nessesary to copy the whole object `product` into memory. Instead, we could directly get `productInfo[productID].seller`. This could save gas. Similar optimization could be applied to `NFTAuction.setMerkleRoot()`, `NFTAuction.setAuctionLocked()` and variable `auctionShareProfit` in function`NFTAuction.finalizeAuction()`. (**Update:** Fixed)
11. Consider remove the check at `NFTAuction.sol`, since the same check on `_storeProfit` has already been done.

Test Results

Test Suite Results

As neither build scripts, nor tests have been provided, no test results could be computed.

Update: During the fix-review process two test files have been provided (`testNFTAuction.js` and `testNFTStore.js`). All 23 of 23 tests were passing.

```
  NFTAuction
owner:0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
buyer:0x70997970C51812dc3A010C7d01b50e0d17dc79C8
buyer2:0x14dC79964da2C08b23698B3D3cc7Ca32193d9955
proxyadm:0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
shareaddr1:0x90F79bf6EB2c4f870365E785982E1f101E93b906
shareaddr2:0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65
shareaddr3:0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc
storeaddr:0x976EA74026E726554dB657fA54763abd0C3a0aa9
selleraddr:0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f
    Init and launch auction
NFTAuction deployed at:0x5FbDB2315678afecb367f032d93F642f64180aa3
MyProxy deployed at:0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
NFTinstance deployed at:0x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0
       ✓ test initialize (58ms)
LAUNCH_ROLE:0xf17a52ef07e0bf6004487e8a911da5d0c2069acda44a7d6c0664442b85327a9e
       ✓ test launch auction fail - before grant LAUNCH_ROLE (116ms)
LAUNCH_ROLE:0xf17a52ef07e0bf6004487e8a911da5d0c2069acda44a7d6c0664442b85327a9e
       ✓ test grantRole (43ms)
       ✓ test launch auction (71ms)
       ✓ test batch launch auction (188ms)
    Bidding test
NFTAuction deployed at:0x0165878A594ca255338adfa4d48449f69242Eb8F
MyProxy deployed at:0xa513E6E4b8f2a923D98304ec87F64353C4D5C853
NFTinstance deployed at:0x2279B7A0a67DB372996a5FaB50D91eAA73d2eBe6
       ✓ test 1st bid (39ms)
NFTAuction deployed at:0xA51c1fc2f0D1a1b8494Ed1FE312d7C3a78Ed91C0
MyProxy deployed at:0x0DCd1Bf9A1b36cE34237eEaFef220932846BCD82
NFTinstance deployed at:0x9A676e781A523b5d0C0e43731313A708CB607508
       ✓ test bid fail - isn't higner than start price
NFTAuction deployed at:0x68B1D87F95878fE05B998F19b66F4baba5De1aed
MyProxy deployed at:0x3Aa5ebB10DC797CAC828524e59A333d0A371443c
NFTinstance deployed at:0xc6e7DF5E7b4f2A278906862b61205850344D4e7d
       ✓ test bid fail - isn't higner than current bid price (44ms)
NFTAuction deployed at:0xa85233C63b9Ee964Add6F2cffe00Fd84eb32338f
MyProxy deployed at:0x4A679253410272dd5232B3Ff7cF5dbB88f295319
NFTinstance deployed at:0x7a2088a1bFc9d81c55368AE168C2C02570cB814F
       ✓ test bid fail - price limitation (48ms)
NFTAuction deployed at:0xE6E340D132b5f46d1e472DebcD681B2aBc16e57E
MyProxy deployed at:0xc3e53F4d16Ae77Db1c982e75a937B9f60FE63690
NFTinstance deployed at:0x84eA74d481Ee0A5332c457a4d796187F6Ba67fEB
       ✓ test bid - no price limitation (57ms)
NFTAuction deployed at:0x851356ae760d987E095750cCeb3bC6014560891C
MyProxy deployed at:0xf5059a5D33d5853360D16C683c16e67980206f36
NFTinstance deployed at:0x95401dc811bb5740090279Ba06cfA8fcF6113778
       ✓ test bid fail - ended auction
NFTAuction deployed at:0x99bbA657f2BbC93c02D617f8bA121cB8Fc104Acf
MyProxy deployed at:0x0E801D84Fa97b50751Dbf25036d067dCf18858bF
NFTinstance deployed at:0x8f86403A4DE0BB5791fa46B8e795C547942fE4Cf
       ✓ test bid fail - canceled auction
NFTAuction deployed at:0x4c5859f0F772848b2D91F1D83E2Fe57935348029
MyProxy deployed at:0x1291Be112d480055DaFd8a610b7d1e203891C274
NFTinstance deployed at:0x5f3f1dBD7B74C6B46e8c44f98792A1dAf8d69154
       ✓ test bid fail - locked auction
    Finalize test
NFTAuction deployed at:0x7969c5eD335650692Bc04293B07F5BF2e7A673C0
MyProxy deployed at:0x7bc06c482DEAd17c0e297aFbC32f6e63d3846650
NFTinstance deployed at:0xc351628EB244ec633d5f21fBD6621e1a683B1181
       ✓ test finalize - check profits and owner (78ms)
NFTAuction deployed at:0x162A433068F51e18b7d13932F27e66a3f99E6890
MyProxy deployed at:0x922D6956C99E12DFeB3224DEA977D0939758A1Fe
NFTinstance deployed at:0x5081a39b8A5f0E35a8D959395a630b68B74Dd30f
       ✓ test finalize fail - not ended
NFTAuction deployed at:0x21dF544947ba3E8b3c32561399E88B52Dc8b2823
MyProxy deployed at:0x2E2Ed0Cfd3AD2f1d34481277b3204d807Ca2F8c2
NFTinstance deployed at:0xD8a5a9b31c3C0232E196d518E89Fd8bF83AcAd43
       ✓ test finalize fail - duplicate


  NFTStore
owner:0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
buyer:0x70997970C51812dc3A010C7d01b50e0d17dc79C8
buyer2:0x14dC79964da2C08b23698B3D3cc7Ca32193d9955
proxyadm:0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
shareaddr1:0x90F79bf6EB2c4f870365E785982E1f101E93b906
shareaddr2:0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65
shareaddr3:0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc
storeaddr:0x976EA74026E726554dB657fA54763abd0C3a0aa9
NFTStore deployed at:0x202CCe504e04bEd6fC0521238dDf04Bc9E8E15aB
MyProxy deployed at:0xf4B146FbA71F41E0592668ffbF264F1D186b2Ca8
NFTinstance deployed at:0x172076E0166D1F9Cc711C77Adf8488051744980C
     ✓ test initialize
     ✓ test launch product (45ms)
     ✓ test buy (46ms)
     ✓ test to buy a sold item
     ✓ test Product Locked/Unlocked (65ms)
     ✓ test share profit (44ms)
     ✓ test cancel product


  23 passing (4s)
```

Code Coverage

As neither build scripts, nor tests have been provided, no coverage could be computed.

Update: During the fix-review process two test files have been provided (`testNFTAuction.js` and `testNFTStore.js`). However, the provided tests achieve only 58% statement coverage and 40% branch coverage. Both of which are way below the recommended minimum of 90%, leaving a lot of code completely untested, potentially hiding further functional/logical issues. We highly recommend adding more tests to achieve at least 90% coverage (and ideally more), before rolling the code out in production.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	58.31	39.56	55.56	59.94	
ERC721PresetHTC.sol	29.73	15	33.33	29.73	… 116,117,120
IERC721HTC.sol	100	100	100	100	
NFTAuction.sol	64.94	45.1	60	66.85	… 609,610,622
NFTStore.sol	57.58	38.33	62.96	59.12	… 401,402,406
All files	**58.31**	**39.56**	**55.56**	**59.94**	

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

```
d62c3cc9f5059c7e222777849c25b7614658eefbfccefffa70a203db97cbab0c ./contracts/NFTAuction.sol

19ea388de64e8bc545e80d1923590b29fbfd1b2eccf38e7aee0be15c749bcba1 ./contracts/NFTStore.sol

34889d8891dbd75d48e610a45882a6d15961aad0808cc3c3671d47effc952815 ./contracts/MyProxy.sol

12af8be738cb82755b7f6b0831c7cb5f4d3b8dc45d7e168782bc5bfe03e0f50b ./contracts/ERC721PresetHTC.sol

29048d1930ab85316e73c20b2714e947a1964080ecd51f6d8155a27f60d9661c ./contracts/IERC721HTC.sol
```

Tests

```
c28597af4e5697bf5d057e08d0d8e7a29000d190651e09f15bfa647b628ac900 ./test/testNFTStore.js

f196021b2b61d1b67f0020ea8f0cb5bf2ea8585bca4fd03ff747d1cce24c6c71 ./test/testNFTAuction.js
```

Changelog

- 2022-08-05 - Initial report
- 2022-09-02 - Final report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.