**Quantstamp** Security Assessment Certificate

December 2nd 2022 — Quantstamp Verified

# Clubhouse Contracts

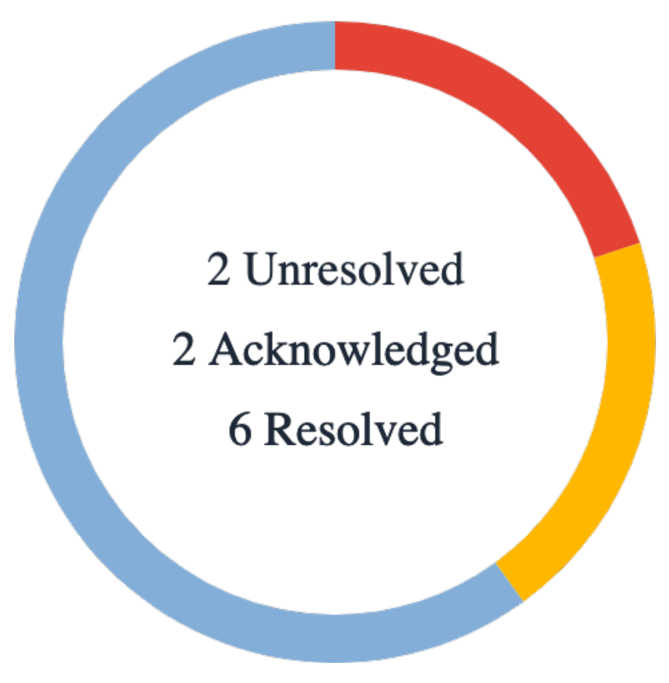
This audit report was prepared by Quantstamp, the leader in blockchain security.

## Executive Summary

| | |
|---|---|
| Type | NFT |
| Auditors | Rabib Islam, Research Engineer<br>Zeeshan Meghji, Auditing Engineer<br>Hytham Farah, Auditing Engineer II |
| Timeline | 2022-10-31 through 2022-12-02 |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review |
| Specification | README.md |
| Documentation Quality | Medium |
| Test Quality | Low |
| Source Code | |

| Repository | Commit |
|---|---|
| https://github.com/alphaexplorationco/clubhouse_contracts/ | 8c164e2 initial audit |

| | |
|---|---|
| Total Issues | **10** (6 Resolved) |
| High Risk Issues | **1** (0 Resolved) |
| Medium Risk Issues | **2** (2 Resolved) |
| Low Risk Issues | **2** (2 Resolved) |
| Informational Risk Issues | **5** (2 Resolved) |
| Undetermined Risk Issues | 0 (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

**Initial Audit:** The aim of these contracts is to construct a process to deploy NFT contracts that issue non-transferable tokens which enable holders to access functions within an application. The project uses a meta-transaction setup with OpenZeppelin Defender. The client wishes to ensure that the setup does not introduce any attack vectors.

We found a number of issues in the project. Importantly, implementing token non-transferability is a hard problem and simple attempts at implementing it can be easily exploited. Moreover, the use of the OpenZeppelin `MinimalForwarder` contract is presented as an issue; as its documentation states, it is not meant for production use. Some low- and informational-severity issues are discussed as well. In addition, we report issues we have found relating to code documentation and best practices. Finally, we recommend the implementation of additional tests.

**Update 1:** Most of the issues were addressed. Notably, QSP-1 was acknowledged, and QSP-6 was left unresolved due to the introduction of a new variable. Moreover, a new informational-severity issue was discovered. We recommend fully addressing the unresolved issues before deployment.

**Update 2:** QSP-6 and QSP-10 were fixed.

| ID | Description | Severity | Status |
|---|---|---|---|
| QSP-1 | Membership Token Ownership Can Be Transferred Using Smart Contract Wallets | High | Acknowledged |
| QSP-2 | Ether Sent to Forwarder May Get Stuck in the Contract | Medium | Fixed |
| QSP-3 | Missing Input Validation | Low | Mitigated |
| QSP-4 | Ownership Can Be Renounced | Low | Fixed |
| QSP-5 | Application Monitoring Can Be Improved by Emitting More Events | Informational | Fixed |
| QSP-6 | Incorrect/Missing Visibility | Informational | Unresolved |
| QSP-7 | EOA Ownership of `UpgradeableBeacon` and `MembershipERC721` | Informational | Fixed |
| QSP-8 | Privileged Roles and Ownership | Informational | Acknowledged |
| QSP-9 | Anyone Can Create a Collection | Medium | Fixed |
| QSP-10 | Clone-and-Own | Informational | Unresolved |

## 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 v0.8.3

Steps taken to run the tools:

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

## Findings

### QSP-1 Membership Token Ownership Can Be Transferred Using Smart Contract Wallets

**Severity:** *High Risk*

**Status:** Acknowledged

**File(s) affected:** `MembershipERC721.sol`

**Description:** The `MembershipERC721` contract includes a `_beforeTokenTransfer()` function that disables transferring of tokens after they are minted, as their ownership confers the ability to access features in an application.
Smart contract wallets (SCWs) are contracts that are designed to custody funds. As contracts, they typically include functions that enable the calling of other smart contracts by the owner, as well as validation that the messages being passed are authorized by the owners.
However, single-owner SCWs can have their ownership (and the ownership of the custodied funds) transferred, and even sold, on-chain via the tokenization of ownership as an NFT (i.e. the SCW checks that the signing party holds an NFT).

**Recommendation:** Implementing token non-transferability is a hard problem that has been discussed at length. However, given that the token owner is designated by contract owner, it makes sense to implement a feature whereby the contract owner can burn the token in case of abuse. In particular, this would require monitoring whether the token-holding addresses are smart contracts (using e.g. `EXTCODESIZE`) and having the contract owner burn a token if its owner's code size is greater than zero. However, this would not prevent momentary smart contract ownership of the token.
Consider the problem of implementing non-transferability and remedy the issue accordingly.

**Update:** The client stated:

> We do not offer a way to connect smart contract wallets to the clubhouse app at the moment, so the risk of transferability via SCWs is low. Additionally, we have added a "transferable" option to enable transferability for some contract instances.

Note that the implementation of the "transferable" option is considered a feature addition and is therefore outside the scope of the audit.

### QSP-2 Ether Sent to Forwarder May Get Stuck in the Contract

**Severity:** *Medium Risk*

**Status:** Fixed

**File(s) affected:** `Forwarder.sol`

**Description:** In a transaction calling `execute()`, `req.value` ether is sent to the `req.to` address. However, in the case that `msg.value` is greater than `req.value`, the excess ether will remain in the `Forwarder` contract.

**Recommendation:** `Forwarder` inherits from `MinimalForwarderUpgradeable`, which contains the issue. As mentioned in the documentation for the contract, `MinimalForwarder` (and its upgradeable variant) is not production-ready. Consider using a different reference contract (e.g. GSN's `Forwarder` contract).

**Update:** `Forwarder` now inherits from OpenGSN's `Forwarder` contract which does not suffer from this issue.

### QSP-3 Missing Input Validation

**Severity:** *Low Risk*

**Status:** Mitigated

**File(s) affected:** `MembershipERC721.sol`

**Related Issue(s):** SWC-123

**Description:** It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. Several functions within the `MembershipERC721` contract lack sufficient validation of their inputs. We have listed all missing checks below:

- `constructor()`: Validate that `_name` and `_symbol` are not empty;
- `safeMint()`: Validate that `expiryTimestamp` is not in the past;
- `updateExpiryTimestamp()`: Validate that `expiryTimestamp` is not in the past.

**Recommendation:** We recommend adding the relevant checks.

**Update:** The constructor now validates that `_name` and `_symbol` are not empty. However, it remains possible to mint an NFT with an `expiryTimestamp` in the past; moreover, the admin retains the ability to expire an NFT by calling `updateExpiryTimestamp()` with an `expiryTimestamp` in the past. It is not clear why an expired NFT should ever be minted; however, the client mentioned that they wish to retain the ability to expire a subscription, though this functionality should be reflected in user-facing documentation.

### QSP-4 Ownership Can Be Renounced

**Severity:** *Low Risk*

**Status:** Fixed

**File(s) affected:** `MembershipERC721.sol`

**Description:** If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed. Both the `MembershipERC721` contract and the `UpgradeableBeacon` contract are ownable contracts and hence suffer from this issue.

**Recommendation:** Double check if this is the intended behavior. If not, then

1. Override the `renounceOwnership()` function within the `MembershipERC721` contract to always revert;
2. Create a new contract which inherits from `UpgradeableBeacon`. Override the `renounceOwnership()` function within the new contract to always revert. Then reference the new contract instead of `UpgradeableBeacon` in the `MembershipERC721Factory` contract.

**Update:** Fixed in commit `5732526101663a2a4c471d0016ff35d78553724a`.

## QSP-5 Application Monitoring Can Be Improved by Emitting More Events

*Severity: Informational*

**Status:** Fixed

**File(s) affected:** `MembershipERC721.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 transitions 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 application management:

- MembershipERC721:
  - `updateExpiryTimestamp()`: Should emit an event indicating that the expiry time of a particular `tokenId` has been updated;
  - `setTrustedForwarder()`: Should emit an event indicating that the contract's forwarder has been changed from one address to another.
  - `safeMint()`: Should emit an event indicating that a new subscription has been given to an address with an `expiryTimestamp`.

**Recommendation:** Consider emitting the events.

**Update:** Fixed in commit `e101eb59d9dbce2895debd832f7cfb7b157958d9`.

## QSP-6 Incorrect/Missing Visibility

*Severity: Informational*

**Status:** Unresolved

**File(s) affected:** `MembershipERC721Factory.sol`

**Related Issue(s):** SWC-100, SWC-108

**Description:** The `beacon` storage field does not have its visibility explicitly declared.

**Recommendation:** The visibility of `beacon` be marked as either `public`, `internal`, or `private` based on whether other or derived contracts should be able to access it.

**Update:** The variable `beacon` has been declared `private`. However, the variable `transferable` in `MembershipERC721` does not have its visibility declared.
**Update 2:** The remainder of the issue was fixed in `038af7d3e5ca1ca0ed4b62b9bdb7d12f20c6e1af` by declaring `transferable private`.

## QSP-7 EOA Ownership of `UpgradeableBeacon` and `MembershipERC721`

*Severity: Informational*

**Status:** Fixed

**File(s) affected:** `MembershipERC721Factory.sol`

**Description:** At the time of creation, both the `UpgradeableBeacon` and `MembershipERC721` contracts can only be owned by "Externally Owned Accounts" (EOAs) as `tx.origin` can only refer to an EOA. An EOA is typically controlled by an individual who possesses the corresponding private key; in such situations, the key can become lost or stolen, leading to the compromise of the protocols the EOA has control over.
In order to mitigate this risk, it is often well-advised to restrict protocol owner roles to multi-signature wallets or secure decentralized governance contracts.
However, in certain cases, EOAs can be used as multi-signature wallets via multi-party computation (MPC).

**Recommendation:** Ensure that the address that owns the `UpgradeableBeacon` and `MembershipERC721` contracts is controlled by a secure process (such as a multi-signature wallet of sufficiently large membership or a decentralized governance process).

**Update:** The client has stated:

> We have changed tx.origin to msg.sender to allow using multisigs as owners. We have also separated the beacon owner and proxy owners so that we can have the proxy be owned by our relayer wallet, while upgrades are controlled by a multisig

## QSP-8 Privileged Roles and Ownership

*Severity: Informational*

**Status:** Acknowledged

**File(s) affected:** `MembershipERC721.sol`, `MembershipERC721Factory.sol`

**Description:** Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. In the case of this project, the `MembershipERC721` and `UpgradeableBeacon` (which is deployed by `MembershipERC721Factory.sol`) contracts have owners. However, as the `UpgradeableBeacon` contract is not in scope, we will not discuss its permissioned functions. The `MembershipERC721` functions with privileged permissions are listed below:

- `mint()` -- mint a token from the collection to any address;
- `updateExpiryTimestamp()` -- update the expiry time of any token in the collection;
- `setTrustedForwarder()` -- reset the trusted forwarder of the contract.

Note also that the project uses upgradeable contracts, which gives the owner the ability to change the implementation contracts when desired. This can leave users unable to rely on the contracts they are interacting with to behave predictably.

**Recommendation:** All special permissions assigned to the contract owners should be clearly documented in the user-facing documentation.

**Update:** The Clubhouse team added code comments to the aforementioned functions. We recommend that the Clubhouse team updates user-facing documentation to explain the privileged permissions mentioned above.

### QSP-9 Anyone Can Create a Collection

**Severity:** *Medium Risk*

**Status:** Fixed

**File(s) affected:** `MembershipERC721Factory.sol`

**Description:** Anyone can call the `buildMembershipERC721Proxy()` function to create a new collection with any name or symbol. Malicious actors may try to spoof already existing collections by using the same name or symbol. It is unclear whether it is desirable for anyone to be able to create a collection or not as the documentation does not indicate this.

**Recommendation:** If every account should have the ability to create a new collection, then nothing needs to be done. Otherwise, access control should be added to restrict access to the `buildMembershipERC721Proxy()` function.

**Update:** The issue was fixed in commit `893496db204ab94f8175198e91a67e3c578b7207`.

### QSP-10 Clone-and-Own

**Severity:** *Informational*

**Status:** Unresolved

**File(s) affected:** `contracts/MembershipERC721Factory.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, with respect to security, 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 commit `5732526101663a2a4c471d0016ff35d78553724a`, the client included a fork of OpenZeppelin's `UpgradeableBeacon` contract, overriding the `renounceOwnership()` function inside the fork.

**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.
In the current case, we recommend using the OpenZeppelin implementation of `UpgradeableBeacon` while overriding `renounceOwnership()` in contracts that inherit it.

**Update:** The issue was fixed in commit `c8667918dd439ffdd115765a659342b7c778b008` by creating a new contract that inherits from `UpgradeableBeacon` and overrides `renounceOwnership()` to make it always revert.

## Automated Analyses

### Slither

All the high and medium severity issues found by Slither were associated with the OpenZeppelin libraries and are false positives. The remainder are false positives as well.

## Code Documentation

- The public interfaces of the contracts are not fully annotated using NatSpec.
- Address the TODO at `MembershipERC721.sol#47`.
- Typo:
  - "timestsamp" at `MemershipERC721.sol#52`.

**Update:** Code documentation issues have been addressed.

## Adherence to Best Practices

- Avoid the use of `tx.origin` as it may become deprecated in the future.
- Instead of storing the `MembershipERC721Factory.beacon` variable as type `UpgradeableBeacon`, it is recommended to store it as type `address` and then wrap it as `UpgradeableBeacon(beacon)` when calling its functions. The reason for this is that it is easier to reason about the storage layout of an address type than a contract type.
- Consider implementing custom errors and using them instead of `require` statements. They are more gas-efficient and allow one to pass in data specific to the error.
- Remove or use the unused import in `MembershipERC721Factory.sol`:
  - `import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";`.
- The `versionRecipient` field within `MembershipERC721` can be made into a constant.
- Several functions can be marked as `external` to save gas:
  - `MembershipERC721.safeMint()`;
  - `MembershipERC721.updateExpiryTimestamp()`;
  - `MembershipERC721.setTrustedForwarder()`;
  - `MembershipERC721Factory.buildMembershipERC721Proxy()`.

**Update:** Best practices issues have been addressed.

## Test Results

**Test Suite Results**

All tests are passing. However, tests are not checking for emitted events, and are not checking some basic cases, e.g. failing on `transfer()`(only `transferFrom()` is checked).

```
Forwarder
  ✓ initializer should call MinimalForwarderUpgradeable initializer
  ✓ transparent proxy upgrades should work (208ms)

Membership NFT Contract
  ✓ constructor should lock initializers (63ms)
  ✓ setUp should set name, symbol, trusted forwarder, and display type correctly
  ✓ safeMint should mint token to address with balance == 0
  ✓ safeMint should revert on mint to address with balance > 0
  ✓ safeMint should revert if called by non-owner address
  ✓ updateExpiryTimestamp and getExpiryTimestamp should set/get expiry timestamp correctly
  ✓ updateExpiryTimestamp should revert if called by non-owner address
  ✓ setTrustedForwarder should update trusted forwarder address when called by owner
  ✓ setTrustedForwarder should revert if called by non-owner address
  ✓ _beforeTokenTransfer should revert on NFT transfer except when burning (74ms)
  ✓ tokenURI should return URI with expiry timestamp and display type as params

Membership NFT Proxy Factory Contract
  ✓ constructor should init beacon with caller as owner
  ✓ buildMembershipERC721Proxy should create proxy if does not exist for social club id
  ✓ getBeacon should return beacon address
  ✓ getImplementation should return implementation contract address
  ✓ beacon upgrade should replace implementation backing proxies (65ms)
  ✓ proxyWasCreatedByFactory should return true for existing proxy and false otherwise


19 passing (2s)
```

## Code Coverage

All branches are covered. However, improvements should be made to achieve 100% coverage across the board.

| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
|---|---|---|---|---|---|
| contracts/ | 96.43 | 100 | 94.12 | 96.97 | |
| Forwarder.sol | 100 | 100 | 100 | 100 | |
| MembershipERC721.sol | 94.74 | 100 | 90.91 | 95.45 | 143 |
| MembershipERC721Factory.sol | 100 | 100 | 100 | 100 | |
| **All files** | **96.43** | **100** | **94.12** | **96.97** | |

## Changelog

- 2022-11-04 - Initial report
- 2022-11-25 - Fix review
- 2022-12-02 - Fix review 2

## About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT