Quantstamp Contract Security Certificate

April 22nd 2020 — Quantstamp Verified

Lendroid Rightshare

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Contracts
Auditors	Alex Murashkin, Senior Software Engineer Sebastian Banescu, Senior Research Engineer Martin Derka, Senior Research Engineer
Timeline	2020-04-09 through 2020-04-21
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	"How it works" page Inline documentation (code comments) and README

Source Code

Repository	Commit
Rightshare-contracts	2bdd99b
Rightshare-contracts	d8960ea
Rightshare-contracts	8e280e3

Changelog

- 2020-04-21 - Diff audit (commit `2bdd99b`)
- 2020-04-21 - Diff audit (commit `d8960ea`)
- 2020-04-16 - Initial report (commit `8e280e3`)

Overall Assessment

Overall, the code is mostly well-written and documented. However, we found two high-severity, one medium-severity, and two low-severity issues. In addition, we made five informational-level findings, as well as proposed several suggestions on improving the documentation and coding practices. The severity of three issues was marked as undetermined due to the lack of the necessary information for an accurate assessment.
Test coverage is high, however, we recommend increasing the branch coverage to as close to 100% as possible. We recommend addressing the issues above before going live.
Update: as of commit `d8960ea`, eight issues were resolved, two - partially resolved, and three issues were acknowledged. Test coverage was brought to 100%.
Update (most recent): as of commit `2bdd99b`, all issues were addressed. Test coverage (both statement and branch coverage) is **100%.**

Total Issues	**13**	(10 Resolved)
High Risk Issues	**2**	(2 Resolved)
Medium Risk Issues	**1**	(1 Resolved)
Low Risk Issues	**2**	(0 Resolved)
Informational Risk Issues	**5**	(5 Resolved)
Undetermined Risk Issues	**3**	(2 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).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.

Summary of Findings

ID	Description	Severity	Status
QSP-1	Addresses of Rights contracts can be changed arbitrarily and repeatedly	High	Resolved
QSP-2	Potential denial-of-service in `isApprovedForAll(...)`	High	Resolved
QSP-3	Missing input validation	Medium	Resolved
QSP-4	Block Timestamp Manipulation	Low	Acknowledged
QSP-5	FRight tokens are transferable	Low	Acknowledged
QSP-6	Unlocked Pragma	Informational	Resolved
QSP-7	Function with missing `return` statement	Informational	Resolved
QSP-8	Ignored return values	Informational	Resolved
QSP-9	Unlocked versions in `package.json`	Informational	Resolved
QSP-10	Potential redundancy of exclusivity (given its equivalence to max supply = 1)	Informational	Resolved
QSP-11	Token versions can be set arbitrarily	Undetermined	Resolved
QSP-12	Maximum supply is decremented when decrementing circulating supply	Undetermined	Acknowledged
QSP-13	Serial numbers are reused upon revocation	Undetermined	Resolved

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:

- Maian
- Truffle
- Ganache
- SolidityCoverage
- Mythril
- Truffle-Flattener
- Securify
- Slither

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
6. Installed the Mythril tool from Pypi: `pip3 install mythril`
7. Ran the Mythril tool on each contract: `myth -x path/to/contract`
8. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
9. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
10. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`
11. Installed the Slither tool: `pip install slither-analyzer`
12. Run Slither from the project directory `slither .`

Assessment

Findings

QSP-1 Addresses of Rights contracts can be changed arbitrarily and repeatedly

Severity: *High Risk*

Status: Resolved

File(s) affected: `RightsDao.sol`

Description: The addresses of any of the fRight or iRight token contracts can be changed at any point in time by the owner via the `set_right` function (at commit `8e280e3`). This would be detrimental to the users who have frozen their base assets because they would not be able to unfreeze them anymore. Similarly, all the issued `iRight` tokens could not be revoked anymore.

Recommendation: Only allow changing the rights contracts when there are no existing `fRight` or `iRight` tokens in the `contracts` mapping in the `RightsDao` contract, or set them once upon initialization (e.g., in the constructor).
Update: the Lendroid team has resolved the issue in `d8960ea` by setting contract addresses via the constructor and disallowing updating the addresses at runtime.

QSP-2 Potential denial-of-service in `isApprovedForAll(...)`

Severity: *High Risk*

Status: Resolved

File(s) affected: `TradeableERC721Token.sol`

Description: The `isApprovedForAll(...)` method on `L70` (commit `8e280e3`) assumes `proxyRegistryAddress` points to an actual registry contract. However, since `proxyRegistryAddress` is not validated in the constructor, it could end up being a zero address, and the call `proxyRegistry.proxies(owner)` on `L80` (commit `8e280e3`) may fail.
This issue is, likely, impactful because the token's `approve(...)` method depends on `isApprovedForAll(...)`.

Recommendation: While it may not be possible to validate a strict adherence to an interface, it is recommended, at least, to validate that `proxyRegistryAddress` is non-zero in the constructor. If a zero-address is actually a valid value, it is recommended to check if `proxyRegistryAddress` is a zero-address, and if it is, to call `super.isApprovedForAll(owner, operator)` directly instead of calling `proxyRegistry.proxies(owner)`.
Update: the Lendroid team has resolved the issue in `d8960ea` by checking that registry address is non-zero in `isApprovedForAll`.
Note: since there is no way to confirm strict adherence to an interface, the owner should be still cautious when setting the proxy address: if the contract does not implement the method `proxies(...)`, the `isApprovedForAll` call is still going to fail.

QSP-3 Missing input validation

Severity: *Medium Risk*

Status: Resolved

File(s) affected: `(Multiple)`

Description: Many locations are missing input parameter validation, for example (line numbers at commit `8e280e3`):

1. `TradeableERC721Token.sol`, `L23`: validating `_proxyRegistryAddress`. If it cannot be a zero address, should check it here. **Fixed** in `d8960ea`: Left as is but the value is now checked upon use.
2. `TradeableERC721Token.sol`, `L31`: ensuring `_to` is a non-zero address. **Fixed** in `d8960ea`.
3. `TradeableERC721Token.sol`, `L71-72`: validating `owner` and `operator`. **Fixed** in `d8960ea`.
4. `RightsDao.sol`, `L74`: validating `addr`. **Fixed** in `d8960ea`.
5. `RightsDao.sol`, `L138`: validating `proxyRegistryAddress`. **Fixed** in `d8960ea`.
6. `RightsDao.sol`, `L158`: `freeze(...)` should also validate `values[0]`, to make sure it's non-zero, and `expiry`, to make sure it is not in the past. **Fixed** in `d8960ea`.
7. `IRight.sol`, `L66`: `revoke(...)` needs to validate that `_tokenId > 0` as the check at `L69` passes for the zero Id. **Fixed** in `d8960ea`.
8. `IRight.sol`, `L52`: `issue(...)` could use more validation for the input parameters: `_to` (non-zero), `_baseAssetAddress` (validity), `_parentId` (existence), `_endTime` (that it is in the future), `_baseAssetId` (validity), `_maxISupply` (validity), `_serialNumber` (validity), `_version` (not exceeding the current version). **Fixed** in `d8960ea`.
9. `IRight.sol`, `L75`: `tokenURI(...)` needs a zero-check for `_tokenId`. **Fixed** in `d8960ea`.
10. `IRight.sol`, `L91`: `parentId(...)` should have a zero check. **Fixed** in `d8960ea`.
11. `IRight.sol`, `L97`: `baseAsset(...)` should have a zero check. **Fixed** in `d8960ea`.
12. `Right.sol`, `L21`: validating `_proxyRegistryAddress` in `setProxyRegistryAddress(...)`. **Fixed** in `d8960ea`.
13. `FRight.sol`, `L136`: `endTimeAndISupplies(...)`: `_tokenId` should have a zero check. **Fixed** in `d8960ea`.
14. `FRight.sol`, `L127`: `isIMintAble(...)`: `_tokenId` should have a zero check. **Fixed** in `d8960ea`.
15. `FRight.sol`, `L76`: `tokenURI(...)` should check if `_tokenId > 0`. **Fixed** in `d8960ea`.
16. `FRight.sol`, `L92`: `incrementCirculatingISupply(...)` should validate `_tokenId > 0` and `_amount > 0`. **Fixed** in `2bdd99b`.
17. `FRight.sol`, `L101`: `decrementCirculatingISupply(...)` should validate `_tokenId > 0` and `_amount > 0`. **Fixed** in `2bdd99b`.
18. `FRight.sol`, `L64`: `unfreeze(...)` check if the token Id is greater than zero, and the address is non-zero. **Fixed** in `2bdd99b`.
19. `FRight.sol`, `L52`: `freeze(...)` missing input validation for most of the parameters. For example, `maxISupply` must be greater than 0, otherwise, minting is not possible. **Fixed** in `d8960ea`.

We assessed the severity of this issue as "Medium" since the impact of a missing input validation could be detrimental: missing validation could expose previously untested and unexpected edge case scenarios.

Recommendation: For addresses, ensuring they are not-zero (i.e., `0x0...`). For integer values, ensuring they are greater than zero where applicable.
Update: the Lendroid team has added input validation in `d8960ea`.

QSP-4 Block Timestamp Manipulation

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `FRight.sol` ,

Description: Projects may rely on block timestamps for various purposes. However, it is important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes.
FRight token expiry logic uses timestamps:

1. `FRight.sol`, `L117` (commit `8e280e3`): `(now >= _meta.endTime)`
2. `FRight.sol`, `L69` (commit `8e280e3`): `(now >= _meta.endTime)`

Recommendation: If possible, using block numbers instead of timestamps, or, alternatively, confirming that the required granularity is not affected by block timestamp manipulation.
Update: the Lendroid team has confirmed that the required granularity is not affected by block timestamp manipulation.

QSP-5 FRight tokens are transferable

Severity: *Low Risk*

Status: Acknowledged

Description: The "How it Works" page says: "As the holder of the land parcel, you keep fRights with you. You can transfer or sell the iRights to anyone." It looks, however, that fRights are actually transferable also, which might be misused. For example, if the fRight token would be transferred by mistake, the new owner may never call `unfreeze` and the original NFT owner would not be able to use their token.

Recommendation: If there is no intention to make fRights transferrable, preventing this at the contract level.
Update: According to the Lendroid team: "FRight is intended to be transferrable, and it is possible for the owner of original NFT to transfer the FRight to another user, which is symbolic of transferring the locked version of the original NFT itself".

QSP-6 Unlocked Pragma

Severity: *Informational*

Status: Resolved

File(s) affected: `All files`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.5.*`. 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: `pragma solidity ^0.5.0;` -> `pragma solidity 0.5.11;`
Update: the Lendroid team has resolved the issue in `d8960ea`.

QSP-7 Function with missing `return` statement

Severity: *Informational*

Status: Resolved

File(s) affected: `Right.sol`

Description: The `setApiBaseUrl` should return `bool` but it does not contain a `return` statement. Moreover, the return value of the calls to this function on `L125` and `L128` in `RightsDao.sol` (commit `8e280e3`) are ignored.

Recommendation: Removing the return value for this function such that it does not return anything.
Update: the Lendroid team has resolved the issue in `d8960ea`.

QSP-8 Ignored return values

Severity: *Informational*

Status: Resolved

File(s) affected: `RightsDao.sol`

Description: Return values of some methods are ignored (line numbers at commit `8e280e3`):

1. The return value of `setProxyRegistryAddress` is ignored for the calls on `L142` and `L145` in `RightsDao.sol`.
2. The return value of `issue` is ignored for the call on `L167` and `L185` in `RightsDao.sol`.
3. The return value of `incrementCirculatingISupply` is ignored for the call on `L186` in `RightsDao.sol`.
4. The return value of `decrementCirculatingISupply` is ignored for the call on `L202` in `RightsDao.sol`.
5. The return value of `revoke` is ignored for the call on `L204` in `RightsDao.sol`.
6. The return value of `unfreeze` is ignored for the call on `L216` in `RightsDao.sol`.

Recommendation: Removing the return value for the function, because they can only return `true`, and otherwise, they revert.
Update: the Lendroid team has resolved the issue in `d8960ea`.

QSP-9 Unlocked versions in `package.json`

Severity: *Informational*

Status: Resolved

File(s) affected: `package.json`

Description: "Unlocked" versions may make build environments not fully reproducible, and also carry the risks of unintentional breaking or introducing security issues. In `package.json` (commit `8e280e3`), the following dependencies have unlocked versions:

```
"truffle": "latest",
"openzeppelin-solidity": "^2.5.0",
"solc": "^0.6.4",
"ganache-cli": "istanbul"
"solidity-coverage": "latest",
"@openzeppelin/test-helpers": "latest"
```

Recommendation: Locking the versions, e.g.: `"truffle": "latest"` -> `"truffle": "5.1.21"`. Also, Ganache-CLI should be updated to the latest version (the one that uses the `Muir Glacier` fork by default).
Update: the Lendroid team has resolved the issue in `d8960ea`.

QSP-10 Potential redundancy of exclusivity (given its equivalence to max supply = 1)

Severity: *Informational*

Status: Resolved

File(s) affected: `RightsDao.sol`, `IRight.sol`

Description: iRight metadata has maximum supply and exclusivity. It seems that `max_supply == 1` should be equivalent to `exclusivity == true`.

Recommendation: While we did not find instances where this invariant is violated, we suggest removing such redundancy from the code, as it is potentially error-prone.
Update: the Lendroid team has removed potentially error-prone redundancies in `d8960ea`, however, kept both the exclusivity and maximum supply as distinct concepts in the code. We recommend removing `isExclusive` completely and interpreting tokens with maximum supply as exclusive, to eliminate any inconsistency by design.
Update: the Lendroid team has addressed this in `2bdd99b`.

QSP-11 Token versions can be set arbitrarily

Severity: *Undetermined*

Status: Resolved

File(s) affected: `RightsDao.sol`

Description: The versions of both `fRight` and `iRight` can be set to any arbitrary value above zero at any point in time (at commit `8e280e3`). Assuming the intention is to only have increasing version numbers, without the possibility to go back to a version number or skip version numbers, this might be an issue.

Recommendation: Only allowing incrementing version numbers.
Update: the Lendroid team has resolved the issue in `d8960ea` by adding functions that allow increments.

QSP-12 Maximum supply is decremented when decrementing circulating supply

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `FRight.sol`

Description: The `maxISupply` is being decremented on `L108` (commit `8e280e3`) inside the `decrementCirculatingISupply` function, which is unexpected.

Recommendation: Making this clear to the end-user via the documentation OR better yet creating another function for setting the `maxISupply` by the `fRight` token owner.
Update: the Lendroid team has acknowledged the issue: "Code has been documented to include this information. Will also educate users about this".

QSP-13 Serial numbers are reused upon revocation

Severity: *Undetermined*

Status: Resolved

File(s) affected: `IRight.sol`, `RightsDao.sol`

Description: It is possible to have iRight tokens with the same `serialNumber` in the following way (at commit `8e280e3`):

1. Owner of fRight token calls the `issue_i` function in `RightsDao.sol` two times, which will issue iRight tokens with serial numbers 1 and 2. This will also set the `circulatingISupply` to 2.
2. The owner of the iRight token with serial number 1 call the `revoke_i` for their iRight token, which means that token is burned and the `circulatingISupply` is set to 1.
3. Owner of fRight token calls the `issue_i` function again, which will issue an iRight token with serial number equal to 2. At this point there are two iRIght tokens having the same serial number.

This would make the `tokenURI` return the same value for these two different tokens.

Recommendation: If the serial number should be unique, then allowing serial numbers to be greater than the maximum supply of iRight tokens (see `L59` in `IRight.sol`, commit `8e280e3`) and always incrementing the serial number, similar to a nonce inside of `RightsDao.sol`.
If the serial number could be re-used, documenting this in the README.
Update: the Lendroid team has resolved the issue in `d8960ea` by removing serial numbers from the code.

Automated Analyses

Maian

Maian found no issues for the given commits.

Mythril

Mythril was unable to finish successfully for the given contracts.

Securify

Securify produced multiple findings of the class "LockedEther", all of which were deemed to be false-positives.

Slither

For the commit `8e280e3`, Slither found the issues with pragma (QSP-6) and ignored return values (QSP-8), flagged naming inconsistencies and use of `public` methods that are not being used internally (both findings were outlined in the `Best Practices` section), use of assembly (which is expected), shadowed declarations (which are expected for method overrides). It has also flagged a lack of initialization of `ERC721._ownedTokensCount`, however, this is deemed to be a non-issue. No additional issues were found for the commits `d8960ea` and `2bdd99b`.

Code Documentation

The code is mostly well-documented, however (line numbers at commit `8e280e3`):

1. Each function should at the minimum have a short description of its purpose, plus a description of its parameters and return value. Several functions are missing such code comments. **Fixed** in `2bdd99b`.
2. Members of `Metadata` structure defined in `IRight.sol` and `FRight.sol` are missing documentation. **Fixed** in `d8960ea`.

In addition:

1. `README.md`: Python- and Vyper- related steps should be removed. **Fixed** in `d8960ea`.
2. `README.md`: it says "contracts have been written in Solidity v.6.0" but it looks like `0.5.x` was used. **Fixed** in `d8960ea`.
3. `README.md`: the name `RightsDao.vy` on `L10`, should be changed to `RightsDao.sol`. **Fixed** in `d8960ea`.
4. `IRight.sol`: `L47`: a typo: “mateadata” -> “metadata”. **Fixed** in `d8960ea`.
5. `RightsDao.sol`: `L209`: the comment “Burn an FRight token for a given FRight token Id, and” doesn’t seem to be accurate. **Fixed** in `d8960ea`.
6. `TradeableERC721Token.sol`: `L28`: the meaning of “tokenURI” is unclear in: “Mints a token to an address with a tokenURI”. **Fixed** in `d8960ea`.
7. `RightDao.sol`: `RIght` -> `Right` on `L117` and `L134`. **Fixed** in `d8960ea`.
8. `TradeableERC721Token.sol`, `L78`: the comment is unclear because nowhere it is specified that `proxyRegistryAddress` is the OpenSea contract. Suggesting to introduce a constant or renaming `proxyRegistryAddress`. **Fixed** in `d8960ea`.
9. `require((now >= _meta.endTime) || (_meta.circulatingISupply == 0), "FRT: token is not unfreezable");` - this makes a token to be unfreezable before the expiry. If this is intended, we suggest documenting this. **Fixed** in `d8960ea` (documented).
10. `IRight.sol`, `L66`: There is no validation of the iRight being revoked, and non-expired iRight could be revoked also. If this is intended, we suggest documenting this. **Fixed** in `d8960ea` (added the comment "The IRight can be revoked at any time." in `RightsDao.sol`).
11. `IRight.sol`, `L56` (commit `d8960ea`): The order of parameters in the comment should be `[parentId, endTime, baseAssetId, version]`. **Fixed** in `2bdd99b`.

Adherence to Best Practices

The code, for the most part, adheres to best practices. However, the following items may require attention (line numbers at commit `8e280e3`):

1. `FRight.sol`,`L68`: `unfreeze(...)` can call `isUnfreezable(...)` to avoid code duplication. **Fixed** in `d8960ea`.
2. `RightsDao.sol`: `L40`, `L42`, `L87`, `L98`, `L161`, `L179`, `L180`, `L182`, `L183`, `L196`, `L201`, `L214`: `require(...)` can provide a message explaining the reason of reverting the transaction in case the underlying condition is `false`. **Fixed** in `d8960ea`.
3. In some cases, setting `_ok = false;` is redundant. A more widely accepted style is the following:
    - Replacing `returns (bool _ok)` with `returns (bool)`
    - Removing `_ok = false;`
    - Replacing `_ok = true;` with `return true;`. **Fixed** in `d8960ea`.
4. Several functions that are declared `public` and never used internally could be declared as `external`:
    - `FRight.sol`, `L52-62`: `freeze(address[2],bool,uint256[4])`
    - `FRight.sol`, `L64-74`: `unfreeze(address,uint256)`
    - `Right.sol`, `L17-19`: `setApiBaseUrl(string)`
    - `Right.sol`, `L21-24`: `setProxyRegistryAddress(address)`

- `IRight.sol`, `L52-64`: `issue(address[2],bool,uint256[6])`
- `IRight.sol`: `L66-73`: `revoke(address,uint256)`. **Fixed** in `d8960ea`.

5. The `isIMintable` function defined on `L127` of `FRight.sol` does not check the value of the `isExclusive` flag in the metadata of the `FRight` token. To save gas one could add a `require` statement after `L130` that would check: `require(!_meta.isExclusive);`. **Fixed** in `d8960ea`.
6. The variable and function naming convention is inconsistent between different files. For example, `FRight.sol` uses Camel case, while `RightsDao.sol` uses Snake case. We recommend using a consistent naming convention across all files. **Fixed** in `d8960ea`.
7. `TradeableERC721.sol`, `L53`: it is recommended to use `SafeMath`, consistently with `L46`. **Fixed** in `d8960ea`.
8. `FRight.sol`, `L96-97`, `L105-108`: arithmetic operations should be simplified and made more consistent:
    - Should use `SafeMath`: `_meta.circulatingISupply += _amount;` -> `_meta.circulatingISupply = _meta.circulatingISupply.add(_amount);`
    - `(_meta.circulatingISupply.sub(_amount) >= 0)` is always truthful: `.sub(...)` will revert if it underflows
    - Should use `SafeMath`: `_meta.circulatingISupply -= _amount;`-> `_meta.circulatingISupply = _meta.circulatingISupply.sub(_amount);`
    - Should use `SafeMath`: `_meta.maxISupply -= _amount;` -> `_meta.maxISupply = maxISupply.sub(_amount);`. **Fixed** in `d8960ea`.
9. `TradeableERC721.sol`, `L56`: Compilation warning at `baseTokenURI()` method: `Warning: Function state mutability can be restricted to pure`. **Acknowledged**. However, there does not seem to be an easy fix for it, therefore, it is better to keep as is.
10. `FRight.sol`, `L178` (commit `d8960ea`): Should `isIMintAblebe` `isIMintable`? **Fixed** in `2bdd99b`.
11. `RightsDai.sol`, `L72,81,93,103,113,131,149,173,192,210,224` (commit `d8960ea`): if a function never returns `false`, the return value is unnecessary: the methods could be returning nothing. **Fixed** in `2bdd99b`.

Test Results

Test Suite Results

All tests pass.

```
Contract: IRight
  constructor
     ✓ deploys with owner
  setApiBaseUrl
     ✓ allows owner to set Api Url (125ms)
  setProxyRegistryAddress
     ✓ allows owner to set Proxy Registry Address (62ms)
  issue : all rights
     ✓ mints iRight token to accounts[1]
     ✓ updates the currentTokenId
     ✓ updates the parentId
     ✓ updates baseAsset (38ms)
  issue : exclusive rights
     ✓ updates the tokenURI (299ms)
  issue : non exclusive rights
     ✓ updates the tokenURI (278ms)
  issue : reverts
     ✓ should fail if to is ZERO_ADDRESS (76ms)
     ✓ should fail if _baseAssetAddress is ZERO_ADDRESS (80ms)
     ✓ should fail if _parentId is 0 (92ms)
     ✓ should fail if _endTime is invalid (89ms)
     ✓ should fail if _baseAssetId is 0 (71ms)
     ✓ should fail if version is 0 (75ms)
  revoke
     ✓ should fail when from address is ZERO_ADDRESS (52ms)
     ✓ should pass when token exists (198ms)
     ✓ should fail for incorrect tokenId (61ms)
  function calls with incorrect tokenId
     ✓ revoke fails (53ms)
     ✓ tokenURI fails (41ms)
     ✓ parentId fails (43ms)
     ✓ baseAsset fails (54ms)

Contract: FRight
  constructor
     ✓ deploys with owner
  setApiBaseUrl
     ✓ allows owner to set Api Url (114ms)
  setProxyRegistryAddress
     ✓ allows owner to set Proxy Registry Address (64ms)
  freeze : all rights
     ✓ mints fRight token to accounts[1]
     ✓ updates the currentTokenId
     ✓ updates isFrozen
     ✓ updates baseAsset
     ✓ updates endTime
     ✓ should decrement CirculatingISupply (162ms)
  freeze : exclusive rights
     ✓ updates the tokenURI (250ms)
     ✓ reverts when IMintAble is called
     ✓ should not increment CirculatingISupply (84ms)
  freeze : non exclusive rights
     ✓ updates the tokenURI (229ms)
```

```
    ✓ IMintAble is true
    ✓ should increment CirculatingISupply (310ms)
freeze : reverts
    ✓ fails when called by non-owner (50ms)
    ✓ fails when base asset address is not a contract (97ms)
    ✓ fails when expiry is invalid (62ms)
    ✓ fails when base asset id is invalid (48ms)
    ✓ fails when version is invalid (64ms)
    ✓ fails when _maxISupply is zero (67ms)
    ✓ fails when called again (176ms)
unfreeze
    ✓ fails when called by non-owner (59ms)
    ✓ should fail for incorrect tokenId (112ms)
    ✓ should pass when circulatingISupply is 0 (359ms)
    ✓ should fail when unfreezable (198ms)
function calls with incorrect tokenId
    ✓ tokenURI fails (47ms)
    ✓ isUnfreezable fails (67ms)
    ✓ isIMintable fails (56ms)
    ✓ baseAsset fails (46ms)
    ✓ endTime fails (43ms)
    ✓ incrementCirculatingISupply fails (154ms)
    ✓ decrementCirculatingISupply fails (156ms)

Contract: RightsDao
  constructor
     ✓ fails when deployed with invalid fRightContractAddress (202ms)
     ✓ fails when deployed with invalid iRightContractAddress (226ms)
     ✓ deploys with owner
     ✓ deploys with whitelistedFreezeActivated set to true
     ✓ deploys with currentFVersion set to 1
     ✓ deploys with currentIVersion set to 1
  deactivateWhitelistedFreeze
     ✓ succeeds only when already activated (236ms)
  activateWhitelistedFreeze
     ✓ succeeds only when already deactivated (205ms)
  toggleWhitelistStatus
     ✓ succeeds only when called by owner (213ms)
     ✓ fails when trying to whitelist ZERO_ADDRESS (56ms)
  incrementCurrentFVersion
     ✓ succeeds only when version > 0 (157ms)
  incrementCurrentIVersion
     ✓ succeeds only when version > 0 (121ms)
  setRightApiBaseUrl
     ✓ allows owner to set api base url of f right (253ms)
     ✓ allows owner to set api base url of i right (158ms)
  setRightProxyRegistry
     ✓ allows owner to set proxy registry of f right (262ms)
     ✓ allows owner to set proxy registry of i right (106ms)
  freeze : exclusive rights
     ✓ fails for incorrect _maxISupply (49ms)
     ✓ fails for incorrect _baseAssetId (306ms)
     ✓ fails for incorrect _endTime (57ms)
     ✓ fails for incorrect f_version (113ms)
     ✓ fails for incorrect i_version (118ms)
     ✓ fails if whitelisted freeze is activated and caller is not whitelisted (171ms)
     ✓ succeeds (308ms)
  issueI
     ✓ works for non exclusive (922ms)
     ✓ fails for exclusive (408ms)
  revokeI
     ✓ fails when tokenId is 0 (56ms)
     ✓ succeeds for non exclusive (1243ms)
  unfreeze
     ✓ succeeds when all i tokens are revoked (1604ms)
  when freeze can be performed only by whitelisted accounts
     ✓ succeeds only when activateWhitelistedFreeze is true and sender is whitelisted (638ms)
  unfreeze and iRevoke after expiry of rights
     ✓ succeeds when all i tokens are revoked (682ms)

Contract: ProxyRegistry
  setProxy
     ✓ fails when proxyContractAddress is invalid (205ms)
     ✓ fails when called by non-owner (118ms)
     ✓ succeeds when proxyContractAddress is valid (222ms)

Contract: TradeableERC721Token
  when ProxyRegistryAddress is ZERO_ADDRESS
    constructor
       ✓ deploys with owner
       ✓ deploys with currentTokenId 0
       ✓ deploys with empty baseTokenURI
    tokenURI
       ✓ should work when tokenId is 0 (104ms)
    isApprovedForAll
       ✓ should just return if owner has authorized operator, without invoking ProxyRegistryAddress (78ms)
  when ProxyRegistryAddress is not ZERO_ADDRESS
    isApprovedForAll
       ✓ should invoke ProxyRegistryAddress and return false
       ✓ should invoke ProxyRegistryAddress and return true (86ms)


95 passing (26s)
```

Code Coverage

For commit `8e280e3`: While test coverage is high - `96.98%` of statements and `84.91%` branches are covered - we still recommend getting close to 100% branch coverage in order to mitigate the existence of any potential functional bugs that could be exploited by attackers. Note: to provide an accurate assessment, we included `Strings.sol` and `TradeableERC721Token.sol` even though they were added to `.solcover.js`.

Some suggestions on improving branch coverage:

- `RightsDao.sol`: `L161`: checking the case when both `whitelisted_freeze_activated` and `is_whitelisted[msg.sender]` are `true`; potentially, covering the "else" paths for the various `require` statements across the file.
- `TradeableERC721Token.sol`: adding coverage for the methods `baseTokenURI`, `tokenURI`, and `isApprovedForAll`, to make sure the methods behave as expected.
- `FRight.sol`: `L105-106`, `L116`, and `L68`: the "else" branches.
- `Strings.sol`: the only missing edge case is `L36` (a zero input).

Update: as of commit `d8960ea`, the branch coverage is 100% as shown below.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
FRight.sol	100	100	100	100	
IRight.sol	100	100	100	100	
Right.sol	100	100	100	100	
RightsDao.sol	100	100	100	100	
Strings.sol	100	100	100	100	
TradeableERC721Token.sol	100	100	100	100	
All files	**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

```
eb7752f9b9e587332789b61d84d88ab3700921e8d077735b6999d3ea28161ca6 ./contracts/FRight.sol

3542c9f78235e6d662ccfcd67620179edd484d25c2011e2adf6d42fd493e94b5 ./contracts/IRight.sol

7354194fe0784d34dbdf287bc4d80a9ace243f79e6684bfef322dd165a7112b5 ./contracts/Migrations.sol

5a6e5db3e83ec1d1025caab563572f8178f480a9c7736a3300519af36d1d952e ./contracts/Right.sol

f74a7977562049fff28ddef93cdb56afc83b771e8e739fc9daeb71aa47f1d901 ./contracts/RightsDao.sol

c42bf6ea7e8ba9afdb7d0f12e0405e69163eea7cd90a5e158fbf34f008e0ec47 ./contracts/Strings.sol

cf7d0a411288717c237b43ed0d0ab832813231d56314ce826791c8d694862082 ./contracts/TradeableERC721Token.sol
```

Tests

```
d7803f3319e84cb2ec371ae4bcd219c951cff9d9c0398222321cb513e9be639f ./test/fRight.js

5c8e6d7986ead65ba27e8a7d4a885bf8af5e97309b7b02fb9918a7a9c1e7ecac ./test/iRight.js

891424943b6c3287ff2fada871dbca161e817bbc9c60f32cf1fa25a9c9ea7d77 ./test/rightsDao.js

830bf434c8b13ae96fb8394e088c70c3fc9ea54db435cec64cda030bde55b563 ./test/tradeableERC721Token.js
```

About Quantstamp

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

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing 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.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.