Quantstamp Contract Security Certificate

June 9th 2020 — Quantstamp Verified

dloop Art Registry Smart Contract

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

Executive Summary

Type	Smart Contracts
Auditors	Alex Murashkin, Senior Software Engineer Sung-Shine Lee, Research Engineer Kevin Feng, Software Engineer
Timeline	2020-05-20 through 2020-06-09
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	A business requirement Google doc [private] A PDF on Dropbox [private]
Documentation Quality	High
Test Quality	High

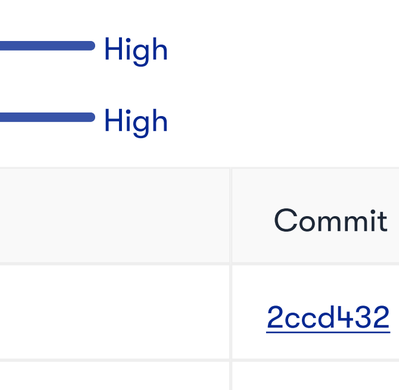
Source Code

Repository	Commit
art-registry	2ccd432
art-registry	42be24b
[privately shared ZIP file with tests]	None

Changelog

- 2020-05-26 - Initial report (42be24b)
- 2020-06-09 - Diff audited: commits 42be24b..2ccd432

Total Issues	**8** (5 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**1** (1 Resolved)
Low Risk Issues	0 (0 Resolved)
Informational Risk Issues	**5** (3 Resolved)
Undetermined Risk Issues	**2** (1 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.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The code is, overall, well-written and accompanied with high-quality tests. However, we identified eight potential issues: one was marked as medium-severity, five - as informational-level, and two - as "undetermined".
There were some mismatches with the specification: the section "Adherence to Specification" provides more details.
The project is lacking in-code documentation. In addition, we suggested several improvements for adherence to best practices.
Update: As of commit `2ccd432`, all issues were resolved. Best practices and Adherence to specification comments were addressed to our satisfaction, and the documentation was updated accordingly.

ID	Description	Severity	Status
QSP-1	Missing input validation	^ Medium	Fixed
QSP-2	Centralization of Power	○ Informational	Acknowledged
QSP-3	Potential denial-of-service or misleading behaviour	○ Informational	Fixed
QSP-4	Missing assertions on the output values	○ Informational	Fixed
QSP-5	Unlocked Pragma	○ Informational	Fixed
QSP-6	Inability to remove or fix incorrect artwork items	○ Informational	Acknowledged
QSP-7	Managed tokens are transferable independently from minters	? Undetermined	Fixed
QSP-8	Block Timestamp Manipulation	? Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

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

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

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Toolset

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

Setup

Tool Setup:

- Maian
- SolidityCoverage
- Mythril
- Slither

Steps taken to run the tools:

1. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
2. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`
5. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
6. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`
7. Installed the Slither tool: `pip install slither-analyzer`
8. Run Slither from the project directory `slither .`

Assessment

Findings

QSP-1 Missing input validation

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `(Multiple)`

Description: Input validation is missing in multiple places (line numbers are for the commit `42be24b`):

1. `DLoopArtwork.sol`, `L22`: The `artworkId` parameter is not checked to be greater than zero, and this violates the requirement: `The artwork id must always be a positive integer`. **Fixed** in `2ccd432`.
2. `DLoopArtwork.sol`, `L25` and `L72`: `dataType` is not currently validated, and therefore, it can be any `bytes32` value. However, `0x0` is used as a special value in the check on `L43` in `DloopMintable.sol`, which suggests that `dataType` should be checked to differ from `0x0` in `DLoopArtwork.sol`. **Fixed** in `2ccd432`.
3. `DloopAdmin.sol`, `L30`: `addAdmin(address account)` should check that `account` is a non-zero address. **Fixed** in `2ccd432`.
4. `DLoopArtwork.sol`: `L26` and `L73`: the `data` parameter is not validated. It could be an empty array, which means it will not contain any hashes. Also, it could be a really long array that may potentially lead to risks of exceeding block gas limits. It is **strongly recommended** to enforce a minimum and a maximum length or, if known, an exact format. Similarly, validation of `data` is missing in `DloopMintable.sol` on `L44` and `L53`. **Fixed** in `2ccd432`.
5. `DloopManagedToken.sol`, `L14`: `_setManaged(...)` does not check if the `tokenId` exists. While in the usage contexts token existence seems to be the case, it is still recommended to add a check into the `_setManaged(...)` method itself. **Fixed** in `2ccd432`.

Recommendation: Validating the input parameters where stated.
Update: the issue has been fixed in `2ccd432`.

QSP-2 Centralization of Power

Severity: *Informational*

Status: Acknowledged

File(s) affected: `(Multiple)`

Description: Smart contracts often have special roles to designate entities with privileges to make modifications to smart contracts. In the context of this project, minter roles assume privileged access such as, ability to add or modify artwork, transfer, or withdraw tokens on behalf of other users. Minters and admins need to be trusted by ordinary users. If privileged members lose their private keys, regular users could suffer from losses or unexpected behavior.

Recommendation: This centralization of power needs to be made clear to the users in the documentation. Users should be made aware of any risks associated with such centralization.
Update: the documentation has been updated accordingly.

QSP-3 Potential denial-of-service or misleading behaviour

Severity: *Informational*

Status: Fixed

File(s) affected: `DloopGovernance.sol`

Description: It is currently possible for an admin to remove all minters using the `removeMinter(...)` method. If no active minters are present, the `onlyMinter` modifier reverts for any Ethereum address, and therefore, the `mintEdition(...)` and `addEditionData(...)` methods are going to revert.
Consider the case when the minter role is enabled (`_minterRoleEnabled` is set to `true`) but there are no active minters. This case could be misleading to both regular users and admins, because when the minter role is enabled, there could be a false expectation that there is at least one active minter. However, there is currently no easy way for admins or users to verify if there are any active minters: `_minterMap` is a map that cannot be traversed, and there is no variable storing the number of minters. So, users would need to know the actual minter address or iterate over past blocks to find `MinterAdded` events.

Recommendation: If the case described in the Description section is not a concern, the code could be left as is, and admins' responsibility to ensure having at least one minter could be documented. If the behavior is concerning, the following mitigation strategies could be considered:

1. Consider disallowing removing all minters when `_minterRoleEnabled` is set to `true`, OR
2. Consider adding `minterCount` to be able to quickly verify how many minters are currently present in the system.

Update: `minterCount` was added in `2ccd432`.

QSP-4 Missing assertions on the output values

Severity: *Informational*

Status: Fixed

File(s) affected: `DloopUtil.sol`

Description: The `splitTokenId(...)` method, `L42-44` (commit `42be24b`): after the bitwise shifts, it is suggested to check if the resulting `artworkId`, `editionNumber`, and `artistProofNumber` are still valid and meet the criteria that were originally imposed in `createTokenId(...)`.

Recommendation: Adding assertions on the resulting `artworkId`, `editionNumber`, and `artistProofNumber`, similar to those in the method `createTokenId(...)`.
Update: the issue has been fixed in `2ccd432`.

QSP-5 Unlocked Pragma

Severity: *Informational*

Status: Fixed

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. For example, replacing `pragma solidity ^0.5.16;` with `pragma solidity 0.5.16;`.
Update: the issue has been fixed in `2ccd432`.

QSP-6 Inability to remove or fix incorrect artwork items

Severity: *Informational*

Status: Acknowledged

File(s) affected: `(Multiple)`

Description: The following scenarios could happen in production:

1. A trusted minter makes a mistake and publishes artwork with valid, but incorrect data. For example, a hash could be computed incorrectly, or the same data being added multiple times.
2. A minter's account is compromised, which leads to fake or unintentional artwork published in the smart contract.

In both of these cases, it is currently impossible to recover: there is no option to correct or delete an artwork.

Recommendation: It is recommended to consider adding admin methods to make corrections to an artwork. However, this may add more centralization to the system. If this is undesired, it is recommended to document the process of error recovery.
Update: the documentation has been updated to reflect the error recovery mechanism.

QSP-7 Managed tokens are transferable independently from minters

Severity: *Undetermined*

Status: Fixed

File(s) affected: `(Multiple)`

Description: When tokens are managed, minters can move tokens on behalf of a regular user, and it is expected. However, in the current implementation, the regular user is also able to move their managed tokens to a new holder, independently from minters (without authorization from minters). Those tokens remain managed after a transfer is complete, but the new token holder may be unaware of the fact that their tokens are still managed, and therefore, the tokens could be taken away from them by the minters.
The specification does not seem to be very clear with regards to this subject. The Google doc says "the platform can move the token internally between managed user accounts". The PDF on Dropbox, while stating that managed tokens are "Only dloop ecosystem controlled / internal tokens", also states that "Technically, the user is able to transfer the tokens too".

Recommendation: It is recommended to check this behavior. If this behavior is unexpected, it is worth overriding the `transfer` and `transferFrom` methods to disallow the transfers of managed tokens for regular users.
If the behavior is expected, and it is documented that **users are aware of the minters' ability to take away the managed tokens that were received from other users**, this finding could be marked as "Acknowledged", and no code change is necessary.

Update: in `2ccd432`, transfer methods were overridden to disallow transferring managed tokens.

QSP-8 Block Timestamp Manipulation

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `DloopWithdraw.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 for their own purposes. If a smart contract relies on a timestamp, it must take this into account.
The logic on `L82` (in `canWithdrawNow()`) can be manipulated by miners, and timestamp precision should not be assumed to be higher than 15 seconds. For example, even for the stated default value of `300` seconds, the actual wait time could potentially vary between, roughly, `275` and `315` seconds.
The smaller the value of withdrawal wait time is, the less reliable it gets: e.g., setting it to `15` seconds (or anything comparable to block duration) could lead to unpredictable behaviour.

Recommendation: It is recommended to re-evaluate the timestamp reliance logic. If high precision is required, it is suggested to switch to using block numbers instead of block timestamps, which are more reliable.

Update: the documentation has been updated to state that the contract does not expect the time precision to be higher than 15s.

Automated Analyses

Maian

Maian did not find any vulnerabilities.

Mythril

Mythril did not find anything within a reasonable time.

Slither

Slither identified a timestamp dependency (`QSP-8`) and unlocked pragma (`QSP-5`). It also made several false-positive or insignificant findings, including the use of assembly and low-level calls (in the OpenZeppelin dependency), suboptimal visibility modifiers, naming inconsistencies, and one instance of `uninitialized-state-variables` which was deemed as a false-positive also.

Adherence to Specification

The code, mostly, adheres to the specification provided in the Google doc, however, we identified several potential discrepancies:

1. `When creating an artwork a user should be required to also specify a “data type” and “data” since every artwork must at the very minimum store the content and metadata hashes required for the above-mentioned provenance guarantee.` While in `DloopArtwork.sol`, in the `createArtwork(...)` method, while the `data` parameter is non-optional, its value is not validated. Therefore, in the code, an empty `data` array that contains none of the hashes would be accepted as valid. It is recommended to add input validation to enforce the presence of the required hashes in the `data` array. **Update:** addressed in `2ccd432`.

2. In `DloopArtwork.sol`, the `createArtwork(...)` method allows for a zero `artworkId` (mentioned in `QSP-1`). This violated the following part of the spec: `The artwork id must always be a positive integer`. **Update:** addressed in `2ccd432`.

3. The implementation does not seem to adhere to the spec item `It must be possible to completely disable the rate-limiting by setting the withdrawal wait time to 0 so that the backend can mint tokens as fast as possible.`: in the code, minting tokens does not depend on the wait time. **Update:** the documentation has been corrected.

4. A note on the item `After the creation of an artwork it must be possible to retrieve the edition and artist proof sizes based on an artwork id. The record “data type” and “data” must also be accessible by artwork id.`: in order to retrieve "data type" and "data" for a given artwork, `getArtworkData(...)` also requires providing the artwork data `index` (`DloopArtwork.sol`, `L92`). Please confirm that this is expected as the `index` parameter is not called out explicitly in this item of the spec. **Update:**

the documentation has been corrected.

5. As requested, a suggestion on implementing the spec item: `While in principle it would have been nice to guarantee that a record for a given "data type" is only added once, we were told by the smart contract developer that such an implementation would be disproportionately complex. In case any of your auditors are aware of a pragmatic way of achieving such a guarantee, we'd be open to any additional input. However, this is not a crucial requirement.` This is possible to achieve in the following way:

    - Maintaining a mapping:

```
mapping(uint64 => mapping(bytes32 => bool)) private dataTypeIsAdded; // artworkId -> dataType -> yes/no
```

    - Setting an entry to `true` when a `dataType` is first added for a given `artworkId`:

```
require(!dataTypeIsAdded[artworkId][dataType], "Data type is already added");
dataTypeIsAdded[artworkId][dataType] = true;
```

   Update: the team decided to not implement the logic and instead documented that in case of duplicate `dataType`s, the latest one is valid.

6. Naming of some methods is inconsistent with the names given in the specification. For example, the specification has the `mint` method as a name for the minting function, however, the implementation uses the name `mintEdition`. **Update:** the documentation has been corrected.

Code Documentation

The project is accompanied by a good-quality specification. However, in the code itself, most methods do not contain developer documentation, except for some of the methods in the `CustomERC721Metadata.sol` file. For all public and external methods, we recommend adding developer documentation stating the semantics of the method, its parameters, and return value. Events and contract state variables should also be documented.

Update: the team has provided a justification for choosing not to document the suggested items.

Adherence to Best Practices

Line numbers are relative to the commit `42be24b`. **Update:** in `2ccd432`, best practices suggestions were addressed where was possible and made sense.

1. `DloopAdmin.sol`, `L3`: the import `ERC721` seems to be unused.
2. It is recommended to use `SafeMath` for all arithmetic operations:
    - `DloopAdmin.sol`: `L33` and `L39`
    - `DLoopArtwork.sol`: `L58` and `L66`
    - `DloopUtil`: `L26` and `L28`
3. `DloopArtwork.sol`, `L44`: `_addTokenToArtwork()` does not seem to associate a token with an artwork. Therefore, the method should be renamed, otherwise, it is misleading.
4. `DloopArtwork.sol`, `L29-30`: the "magic" numbers should be defined as constants.
5. `DloopArtwork.sol`, `L44`: the `_addTokenToArtwork(...)` method is only used in `DloopMintable.sol`, which leads to high coupling with `DloopMintable.sol`. A better practice could be moving the method into `DloopMintable.sol`.
6. `DloopGovernance.sol`: `enableAllMinters(...)` has a `require` statement for `_minterRoleEnabled`, while `disableAllMinters(...)` does not have a similar check.

Test Results

Test Suite Results

All tests are currently passing.

```
  Contract: Dloop Admin
     ✓ Admin - Add/Renounce Admin (2075ms)
     ✓ Admin - Minimum one admin is required (658ms)
     ✓ Admin - Can only be added by admin role (272ms)

  Contract: After Audit Test
     ✓ QSP-1: createArtwork - ArtworkId - Greater Zero (474ms)
     ✓ QSP-1: createArtwork - dataType - Must not be 0x0 (409ms)
     ✓ QSP-1: AddAdmin - account must not be 0x0 (304ms)
     ✓ QSP-1: createArtwork - data.length - data exceeds maximum length (655ms)
     ✓ QSP-1: createArtwork - data.length - data required (241ms)
     ✓ QSP-1: addEditionData - dataType - Must not be 0x0 (760ms)
     ✓ QSP-1: mintEdition - data.length - data exceeds maximum length (1175ms)
     ✓ QSP-1: mintEdition - data.length - data required (670ms)
```

```
    ✓ QSP-3: add or remove Minter - numberOfMinters (340ms)
    ✓ QSP-4: Missing assertions on the output values (284ms)
    ✓ QSP-7: Managed tokens are transferable independently from minters (929ms)
    ✓ Best Practice 6: Disable Minters Require check (552ms)

  Contract: Dloop Artwork
    ✓ Artwork - AddArtworkData (581ms)
    ✓ Artwork - AddArtworkData - edge cases (838ms)
    ✓ Artwork - CreateArtwork - edge cases (1853ms)
    ✓ Artwork - GetArtworkInfo (407ms)

  Contract: All Dloop Events Test
    ✓ Events Test (1155ms)

  Contract: DloopUtil Test
    ✓ to tokenId and verse 1 (274ms)
    ✓ to tokenId and verse 2 (260ms)
    ✓ to tokenId and verse 3 (256ms)
    ✓ check conditions (263ms)

  Contract: Dloop Governance
    ✓ Governance - Enable/Disable Minters (574ms)
    ✓ Governance - Enable/Disable Minters - Edge Cases (500ms)
    ✓ Governance - Disabled Minters (1125ms)
    ✓ Governance - Add/Remove Minters - Edge Cases (582ms)
    ✓ Governance - createArtwork only for Minters (787ms)
    ✓ Governance - addArtworkData only for Minters (838ms)
    ✓ Governance - mintEdition only for Minters (1394ms)
    ✓ Governance - addEditionData only for Minters (1152ms)
    ✓ Governance - managedTransfer only for Minters (1075ms)
    ✓ Governance - withdraw only for Minters (836ms)
    ✓ Governance - withdraw fails for disabled Minters (1071ms)

  Contract: Dloop ManagedTransfer AdvancedTest
    ✓ ManagedTransfer - Minter/Admin can not transfer (890ms)
    ✓ ManagedTransfer - Minter/Admin can not transfer - Disabled Minters (1575ms)
    ✓ ManagedTransfer - DisabledMinters - unmanaged tokens can transfer (1400ms)
    ✓ ManagedTransfer - WithdrawLock - unmanaged tokens can transfer (1925ms)

  Contract: Dloop ManagedTransfer Test
    ✓ ManagedTransfer - managedTransfer (1180ms)
    ✓ ManagedTransfer - Withdraw and ManagedTransfer - fail (936ms)
    ✓ ManagedTransfer - Unauthorized ManagedTransfer - fail (767ms)
    ✓ ManagedTransfer - ManagedTransfer by internal user - fail (945ms)
    ✓ ManagedTransfer - Admin is Unauthorized for ManagedTransfer (749ms)
    ✓ ManagedTransfer - check is managed non existing tokenId  (226ms)

  Contract: Dloop DloopMintable
    ✓ DloopMintable - happy case (518ms)
    ✓ DloopMintable - edge cases (489ms)
    ✓ DloopMintable - setBaseURI (614ms)
    ✓ DloopMintable - AddEditionData (652ms)
    ✓ DloopMintable - MintEdition no dataType (185ms)
    ✓ DloopMintable - AddEditionData - edge cases (730ms)
    ✓ DloopMintable - MintEdition - edge cases (1637ms)
    ✓ DloopMintable - Too many editions and APs (3362ms)
    ✓ DloopMintable - Edition and AP collision (764ms)
    ✓ DloopMintable - AddEditionData - not existing tokenId (360ms)
    ✓ DloopMintable - getEditionDataLength require checks (246ms)
    ✓ DloopMintable - getEditionData require checks (252ms)

  Contract: Dloop SafeTransfer Test
    ✓ SafeTransfer - Mint to smart contract (626ms)
    ✓ SafeTransfer - managedTransfer to smart contract (845ms)
    ✓ SafeTransfer - Withdraw to smart contract (873ms)

  Contract: Dloop Withdraw Test
    ✓ Withdraw - Withdraw checks (737ms)
    ✓ Withdraw - Withdraw token (915ms)
    ✓ Withdraw - Withdraw edge Cases (1334ms)
    ✓ Withdraw - WithdrawLock (1327ms)
    ✓ Withdraw - possible for wait time of 0 (1368ms)
    ✓ Withdraw - LastWithdraw (807ms)
    ✓ Withdraw - Withdraw can not manageTransfer (926ms)

  67 passing (1m)
```

Code Coverage

Test coverage is evaluated as excellent. The test coverage metric is above 99% for statements and 98.72% for branches. The only uncovered branch is in `CustomERC721Metadata.sol`, `L70` (`if (inp == 0) return "0";`), however, covering it is not practical, considering that `tokenId` on `L64` cannot be zero.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	99.39	98.11	100	100	
CustomERC721Metadata.sol	95.83	75	100	100	
DloopAdmin.sol	100	100	100	100	
DloopArtwork.sol	100	100	100	100	
DloopGovernance.sol	100	100	100	100	
DloopManagedToken.sol	100	87.5	100	100	
DloopMintable.sol	100	100	100	100	
DloopToken.sol	100	100	100	100	
DloopUtil.sol	100	100	100	100	
DloopWithdraw.sol	100	100	100	100	
All files	**99.39**	**98.11**	**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

```
544c3d073e625d9f4b478df75cfcbfbb723d2a1a4881f52abfa3ddc1b70dfecb ./contracts/DloopArtwork.sol
95f72f40a955f3ffad6d6ff2f2be15f9379cf4fb83fb3ccb8cba04b495dbb86c ./contracts/DloopToken.sol
7a6137adcfbe337c0808acc46a87801eff19c951c0b915ae56a920367e3cd8b6 ./contracts/DloopWithdraw.sol
e51ec91dddeb93d50a20c67a8857db3a2c664643ea3e0d73eb1bcb1d70115162 ./contracts/DloopMintable.sol
baf2ed7cb0e6acf79537ff5340da9a2efa7c99808f5baba9df853282ba351154 ./contracts/Migrations.sol
78f3d4d0f91dbf2423f52e18da3d99e9f9485383f1b07efe48d2bf2819336bb6 ./contracts/DloopManagedToken.sol
97b639d4faac1bcc2d05717837250b35a2651368b61545eae6f50d8a7b20f7a8 ./contracts/DloopUtil.sol
7af2a0e986989df92a45b8e5d84ed4e97f4486a9dcd13bf64e139c212d5d6e97 ./contracts/DloopGovernance.sol
5d38c094320ea4ab72a72f3173d7dde9f69f3bf6e10e367ed3208bda131efafa ./contracts/CustomERC721Metadata.sol
490af1d68f80e4f292eac4214e6b186a1f24bee17ed9ae6c851c0d578e0fe1f9 ./contracts/DloopAdmin.sol
```

Tests

```
bc4911d0357792e2ef68e5f0b590eeddc2d66347554018b86321178e0f3c96d5 ./test/DloopAllEventsTest.js
1067827107f800c869e9a608d7749f239ab4772751e6634cfe38cd1ce7158d63 ./test/AdminTest.js
e3c1bddce588bc460fdce2a62ea3e3cea5962807b9a286034c1dec9f45b5baf6 ./test/ManagedTransferAdvancedTest.js
d01606dca91f850cb564596f069add52882096bd4c81616774454839ee8bd67a ./test/ManagedTransferTest.js
88bff938be7fecc65a274584027bbcd90217e185fb1f9ac0873bb5912ae3cde8 ./test/MintingTest.js
e1f4b2c5859f9d08b36946382274da22f94ec5ef2b00a4219e9664901baa60ae ./test/DloopUtilTest.js
b57a5453f7411d8a8eced0f57448a32d5693d588cb4b3198951a776a84a7ea98 ./test/SafeTransferTest.js
276af6c290045110c1cf14e8d1f10be30403795306e845a79415fa1774908aba ./test/AfterAuditTest.js
2bd76f65d7692a1d3d32348a58c166b73cfa984002e72efda6b524a579332fe5 ./test/GovernanceTest.js
de3409fed9c9d88d3633fc8984910975c16d50ede657b3d53603f2cfdefc339d ./test/assertThrowsAsync.js
bd0a645622c40b234ffb23bdd812b8af375e6d6732e8428976a29c5eca4c991f ./test/WithdrawalTest.js
280102dc1a403a9b7a4df52c9dc5acf2ad170841b2e9904216f75b72c4001c3f ./test/ArtworkTest.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.