Quantstamp Security Assessment Certificate

January 12th 2021 — Quantstamp Verified

Zora

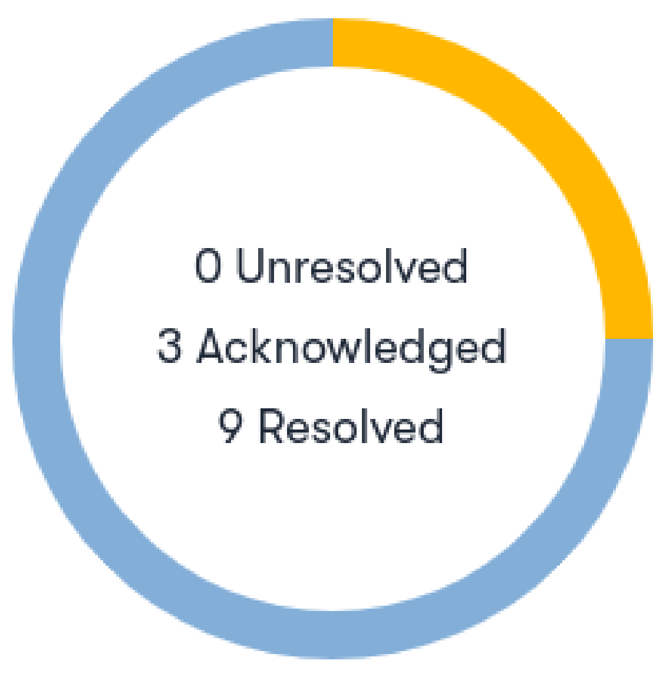
This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	NFT Market
Auditors	Luís Fernando Schultz Xavier da Silveira, Security Consultant Leonardo Passos, Senior Research Engineer Poming Lee, Research Engineer
Timeline	2020-12-14 through 2021-01-11
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	High
Test Quality	Medium
Source Code	

Repository	Commit
ourzora	7964c8c6
ourzora	a12b3fd8

Total Issues	**12**	(9 Resolved)
High Risk Issues	**3**	(2 Resolved)
Medium Risk Issues	**1**	(1 Resolved)
Low Risk Issues	**4**	(3 Resolved)
Informational Risk Issues	**1**	(0 Resolved)
Undetermined Risk Issues	**3**	(3 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

Quantstamp has been able to identify 12 security issues of varying severity: 3 high risk issues, 1 medium risk issue, 4 low risk issues, 1 informational issue and 3 issues of undetermined severity.
Unfortunately it seems some of the issues will require significant code changes to fix.
The scope of the audit are the solidity files in the `contracts/` directory save for the `BaseErc20.sol` file. However, the files `Decimal.sol`, `ERC721.sol`, `ERC721Burnable.sol` and `Math.sol` were only considered to the extent they differ from their clones .
Update 2021-01-06: A re-audit of the codebase at version `v1.0.0` (commit `a12b3fd8c701b949f8171db7c7d6d1214f419b65`) was performed. Two new issues have been included in the report. Overall, there is still only one unresolved issue, with 4 having been fixed, 2 mitigated and 5 acknowledged. The majority of issues in the Best Practices and Code Documentation sections have been addressed as well, though a few remain unresolved.
Update 2021-01-11: QSP-4 has been mitigated, leaving no security issues unresolved.

ID	Description	Severity	Status
QSP-1	Avoidable Sell-on Fees	High	Fixed
QSP-2	ERC721 Transfers are Enabled	High	Acknowledged
QSP-3	NFT Squatting	High	Mitigated
QSP-4	Malicious Bid Shares	Medium	Mitigated
QSP-5	URIs are not Signed for Minting	Low	Mitigated
QSP-6	NFT Vandalism	Low	Mitigated
QSP-7	Seller May Use Psychological Tricks to Induce a Higher Price	Low	Mitigated
QSP-8	Auction System Behaves Differently from a Standard Auction	Low	Acknowledged
QSP-9	Clone-and-Own	Informational	Acknowledged
QSP-10	Incorrect `Media.onlyTokenCreated` Implementation	? Undetermined	Fixed
QSP-11	Re-Entrancy from Untrusted ERC20 Tokens	? Undetermined	Fixed
QSP-12	Incorrect Interface	? Undetermined	Fixed

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 0.6.13

Steps taken to run the tools:

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

Findings

QSP-1 Avoidable Sell-on Fees

Severity: *High Risk*

Status: Fixed

File(s) affected: `Market.sol`

Description: The Zora team is already aware that sell-on fees can be (mostly) avoided and recommends accepting bids only from reputable buyers because of this.
The problem giving rise to this issue is that asks are immediately processed when a bidder makes a suitable bid. Since anyone can make the bid, asks with a high sell-on fee can be accepted by non-reputable parties.
Update 2021-01-06: The issue was fixed by removing sell-on fees from asks.

Exploit Scenario: When Bob makes a bid with a high sell-on fee to Alice and she accepts it, Bob can make a bid to himself with a close to zero amount, which would make him the previous owner, circumventing his duty to pay Alice.

Recommendation: Either disallow sell-on fees on asks or adequately inform users of this issue.

QSP-2 ERC721 Transfers are Enabled

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `Market.sol`, `Media.sol`

Description: Users can transfer tokens via the ERC721 part of the `Media` contract and circumvent the creator fee.
Update 2021-01-06: The Zora team notes that participants engaging in such behavior would have a reputation penalty, which would hurt their prospects as a distribution platform. They also note that the incentive to circumvent the creator fee outside the Zora market is diminished not only by the reputation penalty but also by the available liquidity in that market. They also make the valid point that disabling such transfers would hinder composition with other protocols and the movement of NFTs between a collector's wallets.

Exploit Scenario: Users can negotiate NFTs outside the mechanism of the `Market` contract. They can therefore transfer NFTs without paying either the sell-on fee or the creator share.

Recommendation: Disable ERC721 transfers if possible.

QSP-3 NFT Squatting

Severity: *High Risk*

Status: Mitigated

File(s) affected: `Media.sol`

Description: An attacker can front-run transactions and steal the newly minted NFTs.
Update 2021-01-06: The Zora team presented the following pragmatic mitigation strategy: a user can mint the new media from an unclaimed address and claim such address once the mint has been confirmed. Furthermore, users with stolen tokens and high reputation can denounce such incidents and mint the media with a bit flip instead.

Exploit Scenario: Suppose Alice produces a piece of media and Bob finds out its content hash, either by downloading the media from its URI or by looking at the transaction pool. Bob can then front-run Alice's transaction and claim ownership of the NFT.

Recommendation: We do not have a recommendation at the moment, but significant changes to the protocol design are necessary to fix this issue.

QSP-4 Malicious Bid Shares

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `Decimal.sol`, `Market.sol`

Description: If a bid share is set to a value that is neither a multiple of 2 nor a multiple of 5, then only bid amounts that are a multiple of 10^{20} can be accepted. This may render the token untradable.
Update 2021-01-11: The Zora team suggested the following mitigation tactic. A user with a NFT locked like this can create an ERC20 token for which his balance is 10^{20} and use this token to make a bid himself on the locked token. The bid can then restore the shares to sane values.

Exploit Scenario: The same mechanism described in QSP-6 can be used to set a malicious bid share on a token destined to someone else. Bad bid shares can also be set by unaware users.

Recommendation: To mitigate this, we recommend setting the value of the constant `BASE_POW` in file `Math.sol` to a lower value, say `2`. We also recommend warning against the usage of ERC20 tokens without sufficiently many decimals.

QSP-5 URIs are not Signed for Minting

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `Media.sol`

Description: The digest computed in `mintWithSig` does not depend on the media URIs.
Update 2021-01-06: The issue has been mitigated by paying attention to the URI update events and having the owner fix the URIs if necessary, though a window of time wherein the URIs are defaced still exists.

Exploit Scenario: Whoever has the signature can decide which URIs the minted token will have. Note that this includes anyone observing the transaction pool, so untrusted users can potentially hijack a transaction by having miners include theirs before the original.

Recommendation: Include `data.tokenURI` and `data.metadataURI` in the digest computation.

QSP-6 NFT Vandalism

Severity: Low Risk

Status: Mitigated

File(s) affected: `Market.sol`, `Media.sol`

Description: For a small price, spiteful users may deface NFTs before their target acquires them.
Update 2021-01-06: This is mitigated in a similar manner to QSP-5.

Exploit Scenario: Suppose Alice is selling a NFT for which Bob has a bid. If Charles hates Bob, he can offer a slightly better bid, which will likely be selected by Alice instead of Bob's. He then uses the `updateTokenURI` and `updateTokenMetadataURI` functions of the `Media` contract to deface the token and then proceeds to accept Bob's original bid (for a very slightly lower price) and pass the defaced token to Bob. Unless Bob notices, the URIs will be broken (or worse, contain repulsive content). This could drag the NFT price down or damage Bob's reputation.

Recommendation: One solution to this is to use a nonce-based bid expiry system, expiring all bids for a NFT as its owner accepts such a bid. Another (in our opinion worse) solution is to allow only the creator of the NFT to change its URIs. The creator receives fees on transactions and so has an incentive to keep the NFT in shape.

QSP-7 Seller May Use Psychological Tricks to Induce a Higher Price

Severity: Low Risk

Status: Mitigated

File(s) affected: `Market.sol`, `Media.sol`

Description: A seller (or another colluding party) may place artificial bids on their own auctions to give gullible users the impression their media is worth more than it actually is.

Recommendation: Explain this to users.

QSP-8 Auction System Behaves Differently from a Standard Auction

Severity: Low Risk

Status: Acknowledged

File(s) affected: `Market.sol`, `Media.sol`

Description: The Zora NFT market behaves differently from a standard auction in two main ways:

- There is no time window wherein bids are placed; as soon as a bid meets the ask, it is accepted, potentially resulting in a lower payout to the seller.
- Asks may be modified (reduction, increase, change in currency, insertion, removal), even in response to a bid.

Since the term “auction” is used, some users might misunderstand the nature of the Zora market.

Recommendation: Make sure the documentation informs users of this difference.

QSP-9 Clone-and-Own

Severity: Informational

Status: Acknowledged

File(s) affected: `Decimal.sol`, `Math.sol`, `ERC721.sol`, `ERC721Burnable.sol`

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

Recommendation: Acknowledge the risks, as this seems justified. Also, consider verifying the audit status of the cloned files.

QSP-10 Incorrect `Media.onlyTokenCreated` Implementation

Severity: Undetermined

Status: Fixed

File(s) affected: `Media.sol`

Description: Line 159 should have a strict inequality instead.

QSP-11 Re-Entrancy from Untrusted ERC20 Tokens

Severity: Undetermined

Status: Fixed

File(s) affected: `Market.sol`, `Media.sol`

Description: Using untrusted ERC20 tokens opens up re-entrancy possibilities. Although we have not been able to identify a precise exploit, the logic in contracts `Market` and `Media` is complex enough to prevent us from having complete confidence these do not exist.

Recommendation: Mark all storage modifying functions of contract `Media`with the `nonReentrant` modifier.

QSP-12 Incorrect Interface

Severity: *Undetermined*

Status: Fixed

File(s) affected: `IMedia.sol`

Description: The `mint` function has a different set of parameters.

Automated Analyses

Slither

Slither failed due to a name clash between OpenZeppelin's `Math` library and the one cloned from dYdX. We recommend fixing this issue as Slither is a very useful tool for analyzing smart contracts.

Adherence to Specification

1. The base URI of the `Media` contract cannot be set, but code in that contract indicates it is desired. **Update 2021-01-06:** fixed by removing logic related to the base URI.
2. In contract `Market`, if a bid is made and then removed, its funds may end in a different address.

Code Documentation

1. Make it clear that `BaseErc20.sol` will not be used in production. **Update 2021-01-06:** not fixed.
2. As done for other cloned files, make sure Math.sol has a reference to the file it was clone from. **Update 2021-01-06:** fixed.
3. `Decimal.sol` L21: this is false; see, e.g, new `add` and `sub` functions. **Update 2021-01-06:** fixed by removing these unused functions.
4. Speaking of which, these two functions are not necessary and, furthermore, the precision of their `amount` parameter is unclear from the signature. **Update 2021-01-06:** fixed as the issue above.
5. Write code comments for every function. **Update 2021-01-06:** fixed.
6. It is preferable that code comments use the Natspec format. **Update 2021-01-06:** fixed.
7. `Market.sol` L186: not necessarily equal. **Update 2021-01-06:** fixed.
8. `Market.sol`: given the name `_splitShare`, one would expect it to be private. **Update 2021-01-06:** fixed.
9. `Market.sol` L339,340: it can only be called from the media contract. **Update 2021-01-06:** fixed.
10. `Market.sol` L344: it would be helpful to document which instances you have in mind. **Update 2021-01-06:** wouldn't that case have already been handled with the bid being rejected when it was made?
11. `Media.sol` L470,471: how would one use `balanceOf`? **Update 2021-01-06:** fixed.
12. `Media.sol` L115: what is an "operator query"? **Update 2021-01-06:** fixed.
13. The name `owner` for the parameters of the events of contract `Media` is misleading. Consider renaming them to something akin to `updater`. **Update 2021-01-06:** not fixed.
14. `Market.sol` L32,41: consider replacing "previous" by "current". **Update 2021-01-06:** fixed.
15. `Media.sol` L460,465: consider replacing "non-empty" by "non-zero". **Update 2021-01-06:** fixed.
16. `Media.sol` L69: "sha 256". **Update 2021-01-06:** fixed.
17. `ERC721.sol` L10: "visibiility". **Update 2021-01-06:** not fixed.
18. `Media.sol` L449: "tbe". **Update 2021-01-06:** fixed.
19. `Media.sol` L467: double space. **Update 2021-01-06:** fixed.

Adherence to Best Practices

1. Consider checking that:

    - `mediaContractAddress != address(0)` in `Market.configure`; **Update 2021-01-06:** fixed.
    - `currencyAddress != address(0)` in modifier `onlyTransferAllowanceAndSolvent` of `Market.sol`; **Update 2021-01-06:** fixed.
    - `bid.currency != address(0)` in `Market.setBid`; **Update 2021-01-06:** fixed.
    - `bid.recipient != address(0)` in `Market.setBid`; **Update 2021-01-06:** fixed.
    - `bidder != address(0)` in `Market.removeBid`. **Update 2021-01-06:** not fixed.

2. Make public functions not called within the contracts have external visibility instead in order to save gas. **Update 2021-01-06:** fixed.
3. Make contract `Media` inherit from `IMedia` so the compiler can help check the interface correctness. **Update 2021-01-06:** fixed.
4. Simply returning zero in `Math.sol` L37 is enough. **Update 2021-01-06:** not fixed.
5. In contract `Media`, turn `PERMIT_TYPEHASH` and `MINT_WITH_SIG_TYPEHASH` into constants and give them a visibility modifier. Consider also using the code in the comments above them for their initialization. **Update 2021-01-06:** fixed.
6. Lines 5, 6, 7 and 15 of `Market.sol` are unnecessary. **Update 2021-01-06:** fixed.
7. Remove line 243 of `Market.sol`. **Update 2021-01-06:** fixed.
8. It makes little sense to have the `Decimal` library if its internals are exposed in the `Market.sol` file. **Update 2021-01-06:** not fixed.
9. Is there a point to having `onlyTransferAllowanceAndSolvent` in the `Market` contract given that the transfer would fail down the line if this condition is not met? **Update 2021-01-06:** fixed.
10. `Media.sol`, functions `mint` and `mintWithSig`: modifiers `onlyValidURI` are not necessary. **Update 2021-01-06:** fixed.

Test Results

Test Suite Results

All tests are passing.

```
Market
  #constructor
     ✓ should be able to deploy (2163ms)
  #configure
     ✓ should revert if not called by the owner (467ms)
     ✓ should be callable by the owner (1129ms)
     ✓ should reject if called twice (1398ms)
  #setBidShares
     ✓ should reject if not called by the media address (320ms)
     ✓ should set the bid shares if called by the media address (1207ms)
     ✓ should emit an event when bid shares are updated (1076ms)
     ✓ should reject if the bid shares are invalid (313ms)
  #setAsk
     ✓ should reject if not called by the media address (268ms)
     ✓ should set the ask if called by the media address (2217ms)
     ✓ should emit an event if the ask is updated (2006ms)
     ✓ should reject if the ask is too low (993ms)
     ✓ should reject if the bid shares haven't been set yet (285ms)
  #setBid
     ✓ should revert if not called by the media contract (319ms)
     ✓ should revert if the bidder does not have a high enough allowance for their bidding currency (353ms)
     ✓ should revert if the bidder does not have enough tokens to bid with (1452ms)
     ✓ should revert if the bid currency is 0 address (1842ms)
     ✓ should revert if the bid recipient is 0 address (1869ms)
     ✓ should revert if the bidder bids 0 tokens (1987ms)
     ✓ should accept a valid bid (3286ms)
     ✓ should accept a valid bid larger than the min bid (3147ms)
     ✓ should refund the original bid if the bidder bids again (4746ms)
     ✓ should emit a bid event (2866ms)

Media
  #constructor
     ✓ should be able to deploy (3351ms)
  #mint
     ✓ should mint a token (8045ms)
     ✓ should revert if an empty content hash is specified (254ms)
     ✓ should revert if the content hash already exists for a created token (6885ms)
     ✓ should revert if the metadataHash is empty (269ms)
     ✓ should revert if the tokenURI is empty (238ms)
     ✓ should revert if the metadataURI is empty (239ms)
     ✓ should not be able to mint a token with bid shares summing to less than 100 (596ms)
     ✓ should not be able to mint a token with bid shares summing to greater than 100 (513ms)
  #mintWithSig
     ✓ should mint a token for a given creator with a valid signature (9344ms)
     ✓ should not mint a token for a different creator (533ms)
     ✓ should not mint a token for a different contentHash (540ms)
     ✓ should not mint a token for a different metadataHash (529ms)
     ✓ should not mint a token for a different creator bid share (582ms)
     ✓ should not mint a token with an invalid deadline (490ms)
  #setAsk
     ✓ should set the ask (4075ms)
     ✓ should reject if the ask is 0 (267ms)
     ✓ should reject if the ask amount is invalid and cannot be split (323ms)
  #removeAsk
     ✓ should remove the ask (4236ms)
     ✓ should emit an Ask Removed event (4739ms)
     ✓ should not be callable by anyone that is not owner or approved (3463ms)
  #setBid
     ✓ should revert if the token bidder does not have a high enough allowance for their bidding currency (398ms)
     ✓ should revert if the token bidder does not have a high enough balance for their bidding currency (1021ms)
     ✓ should set a bid (2385ms)
     ✓ should automatically transfer the token if the ask is set (36329ms)
     ✓ should refund a bid if one already exists for the bidder (37991ms)
  #removeBid
     ✓ should revert if the bidder has not placed a bid (280ms)
     ✓ should revert if the tokenId has not yet ben created (206ms)
     ✓ should remove a bid and refund the bidder (1261ms)
     ✓ should not be able to remove a bid twice (1198ms)
     ✓ should remove a bid, even if the token is burned (3316ms)
  #acceptBid
     ✓ should accept a bid (10859ms)
     ✓ should emit an event if the bid is accepted (10746ms)
     ✓ should revert if not called by the owner (261ms)
     ✓ should revert if a non-existent bid is accepted (275ms)
     ✓ should revert if an invalid bid is accepted (7529ms)
  #transfer
     ✓ should remove the ask after a transfer (5102ms)
  #burn
     ✓ should revert when the caller is the owner, but not creator (1216ms)
     ✓ should revert when the caller is approved, but the owner is not the creator (1631ms)
     ✓ should revert when the caller is not the owner or a creator (218ms)
     ✓ should revert if the token id does not exist (198ms)
     ✓ should clear approvals, set remove owner, but maintain tokenURI and contentHash when the owner is creator and caller (2145ms)
     ✓ should clear approvals, set remove owner, but maintain tokenURI and contentHash when the owner is creator and caller is approved (2247ms)
  #updateTokenURI
     ✓ should revert if the token does not exist (197ms)
     ✓ should revert if the caller is not the owner of the token and does not have approval (227ms)
     ✓ should revert if the uri is empty string (224ms)
     ✓ should revert if the token has been burned (9290ms)
     ✓ should set the tokenURI to the URI passed if the msg.sender is the owner (2291ms)
     ✓ should set the tokenURI to the URI passed if the msg.sender is approved (3211ms)
  #updateMetadataURI
```

```
      ✓ should revert if the token does not exist (215ms)
      ✓ should revert if the caller is not the owner of the token or approved (221ms)
      ✓ should revert if the uri is empty string (225ms)
      ✓ should revert if the token has been burned (9164ms)
      ✓ should set the tokenMetadataURI to the URI passed if msg.sender is the owner (2476ms)
      ✓ should set the tokenMetadataURI to the URI passed if the msg.sender is approved (3251ms)
    #permit
      ✓ should allow a wallet to set themselves to approved with a valid signature (3746ms)
      ✓ should not allow a wallet to set themselves to approved with an invalid signature (622ms)
    #supportsInterface
      ✓ should return true to supporting new metadata interface (126ms)
      ✓ should return false to supporting the old metadata interface (124ms)
    #revokeApproval
      ✓ should revert if the caller is the owner (212ms)
      ✓ should revert if the caller is the creator (195ms)
      ✓ should revert if the caller is neither owner, creator, or approver (212ms)
      ✓ should revoke the approval for token id if caller is approved address (2448ms)


  86 passing (23m)
```

Code Coverage

Code coverage is not supported. We highly recommend the Zora team to enable that feature.

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

```
8080d655a2ee87d993a9c83cc04d8b13ee8ce918bf7e8b03974b7bef1652ae68 ./contracts/Market.sol
afbda48e2c23ad81a00233874b3ba36ccaa94be1cd1646e0780a402031e788ee ./contracts/ERC721.sol
505d49e11a0d867c764a3cee864f7154ce67c5b3e5a9c5a2574d3a50fe135ed0 ./contracts/Math.sol
b451932d48d5cb9bf2faf5eaf0ec957c24691023bebec87a38023eeca3c7b32e ./contracts/Decimal.sol
70ab8f12fde9958c2dfd019ac483f67fb85b2c99bec7ca75cff56cecc34f278b ./contracts/Media.sol
fa68f02c18b10f6fce6e27ab12e4a044256681826356e5f1e8cf8e39946d7fb2 ./contracts/ERC721Burnable.sol
a168b934594131f0a8aa41ddd8112232ebe1697c994056280b9c7178e3163e81 ./contracts/interfaces/IMarket.sol
cbd6c06a3e4c4475e78a3c8765196e12fac39b316dee4a52ea2cd7f34c44cf43 ./contracts/interfaces/IMedia.sol
```

Tests

```
c7c1bde793f1cbcaf2fb86a907fc642024ada472c7c58a7a2cdaef2158341cd9 ./test/Media.test.ts
b0907dfd1af886c22f6a5a10b5583fcd05642c5994f0159be252115d1703082a ./test/Market.test.ts
f15c07cbb9a17c14d9c4211223945d9bbb941a947e666a19b165b1ff56c0e259 ./test/utils.ts
```

Changelog

- 2020-12-18 - Initial report
- 2021-01-11 - Updated report

About Quantstamp

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

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

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

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