STAUBLI
SOFTWARE
SOLUTIONS

Smart Contract Code Review and Audit Report

January 28th, 2023

—

Created for: VAST.app

Roger Staubli
Staubli-Software-Solutions
St.Niklausengasse 19
6010 Kriens, Switzerland

STAUBLI
SOFTWARE
SOLUTIONS

Document

Name	Smart Contract Code Review and Audit Report
Carried out by	Roger Staubli | Founder Staubli-Software-Solutions
Language	Solidity
Methods	Best Practice Review, Manual Code inspection, Automated Code inspection
Repository	https://github.com/vast-app/smart-contracts
Commit	Initial: 180bb1e2b02fa2627d2db91f13274abb9c1ba53a
Technical Documentation	Yes: Provided Whitepaper Draft
Unit Tests	Yes
Timeline	16.01.2023 – 28.01.2023
Changelog	27.01.2023 - Initial Audit 28.01.2023 - Final Audit

STAUBLI SOFTWARE SOLUTIONS

STAUBLI
SOFTWARE
SOLUTIONS

Executive Summary

The initial audit resulted in 0 critical severity issues, 0 medium severity issues, and 3 low severity issues. In addition, 3 informational suggestions were provided. All issues and suggestions were resolved for the final audit.

VAST.app is a decentralized marketplace that allows users to buy and sell digital media, such as videos and images. The platform utilizes blockchain technology to ensure secure transactions and copyright protection for digital media. The marketplace offers various options for buying and selling digital content, including auctions, direct sales, public minting, and gifting. Additionally, the platform supports different commission types for creators, sellers, and auction organizers. It offers two distinct markets for trading digital assets: an external market for trading external digital assets and an internal market for trading and minting self-created ERC721 tokens. The ERC721 contract is also extended to allow predefined creators to create collections and trade them on the internal market.

Overall, the contracts are well programmed, with a clear separation of concerns and respect for Smart Contract best practices. Also, the test coverage is exceptionally high, with thorough coverage of both normal and edge cases.

Scope

The audit contained all Smart Contracts, tests, and deployment scripts from the specified commit described on the second page. Well-known dependencies like OpenZeppelin implementations were out of scope.

The following Smart Contract files were reviewed:

cf1116872d006beb0a05d96ad0d07c8dd3ff975b	contracts/ExternalMarket/ExternalAuction.sol
ccfabfc8078f40123b2ff26319d1a368c41e204b	contracts/ExternalMarket/ExternalMarket.sol
2878af129ed2d98f1f5fd9dfa17877b7591c9200	contracts/ExternalMarket/ManagedExternalMarket.sol
e0c3017d777c720b2f1413e0c2cda950fa68ebbb	contracts/Market/Auction.sol
d196a10c010d15b0cab2b147ad9a65fa3c0f6e70	contracts/Market/ManagedMarket.sol
0061c27325961c42103df92c722bd0498621ff62	contracts/Market/Market.sol
ab7ea28924d332b1556ccef6f2c4d4c03a38799c	contracts/Market/PublicMint.sol
6c3caab55aa11f64eb46d1e2501fbb88cd454a42	contracts/SignedApproval.sol
dd1a66ee254d421d4a3bae63cdfc0e2ad1dc2c4c	contracts/Token/DigitalMediaManager.sol
50a00a9f57639dc3ca72b61a7ec7d6c93500e6f1	contracts/Token/DigitalMediaReleaseManager.sol
410ca97641e9ca2b71fec3141994675882d2fceb	contracts/Token/DigitalMediaToken.sol
3bcc345ad922ce830a1f386f796c47b5630ce118	contracts/Token/ERC721Token.sol
44116e20f9ba6e6fb4bc8f1332502f81caaaeb26	contracts/Token/ManagedToken.sol

The following tests and deployment scripts were reviewed:

36ea3186cad3aa0a0ee1452ea47049a10c7fc75b	scripts/deploy-external-market.ts
f540988a54d9e55af72b09d71826a2c4f84e2794	scripts/deploy.ts
b2f84ffdafffdc14fd8c1c16b676254cc8da8e08	scripts/transfer-ownership.ts
7cafb5c5e3935b51471a3114ec75995537b02fd9	test/Auction.ts
358cbaed82d80fbd37ba8c23089d431b8fbf6df5	test/DigitalMediaManager.ts
d2a92605579bcb86915768dad35e87d94ba5452f	test/DigitalMediaReleaseManager.ts
7605c9b0c0d9079892f50acc27cd2a7598a4d4e7	test/DigitalMediaToken.ts
e3ffe8ab257960b49d3ee9f27c6c00a431730d79	test/ERC721Token.ts
d19a646a4ee983c24f77910f58013a4eb80037d0	test/ExternalAuction.ts
0292375eeef39b5ddc7fcda89bb8f1652be3b33b	test/ExternalMarket.ts
14340bb12f6f75545d3fbfca0446d8978aaae893	test/ManagedExternalMarket.ts
9c92f5d4147b9934a353b494862e1b41fda56bdd	test/ManagedMarket.ts
ef828204eb130ab80bbedc420f9c774c1574d75d	test/ManagedToken.ts
e17e55ae3e90182bf012643600d218a0a4fd6e5b	test/Market.ts
6e43da378257a9a904b541940f5af3e47c53b8ad	test/PublicMint.ts
60e128c4f13dcd2f83292e760cc0712022b58db1	test/SignedApproval.ts

System Description

The Vast.app contracts is a set of contracts that provide a marketplace for digital media and an ERC721 Token which is used to create new unique or collections of NFT's. The following section briefly describes the functionalities of the individual contracts. It is important to note that all contracts are upgradeable using OpenZeppelin upgrades.

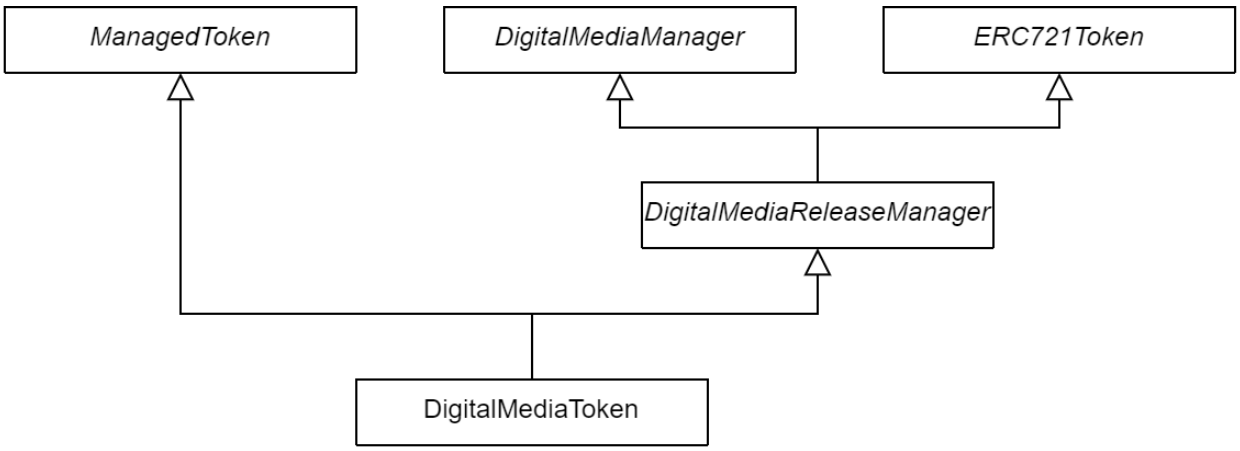
DigitalMediaToken.sol

The DigitalMediaToken.sol is a smart contract that implements the ERC721 token standard for digital media collections. This contract allows creators to manage and create digital media collections, including creating new digital media, minting new releases, updating collection metadata, and burning digital media.

The contract utilizes the DigitalMediaManager.sol contract, which stores the properties of each digital media item, such as the creator's address, the total supply, the release count, the metadata path, and the revealed metadata path. Once a digital media is created, the minting of releases can occur, which are unique ERC721 tokens that are mapped back to their corresponding digital media.

Releases can be minted by the media creator or approved public minters, such as a market contract. The collection metadata can be updated by the media creator and approved system accounts, allowing the media to be switched to revealed and the revealed metadata path to be

set. Tokens can be burned by the owner or with the approval of another spender. Additionally, Digital media can be burned by the media creator by setting the release count to the total supply, preventing any further minting of the underlying token.



Internal functionality is distributed across different abstract contracts. The ManagedToken.sol handles authorization; the DigitalMediaManager is responsible for managing digital media objects; the ERC721Token.sol contract contains the actual ERC721 functions, and the DigitalMediaReleaseManager.sol cares about the mapping between digital media and the actual ERC721 released tokens.

ManagedToken.sol

The abstract ManagedToken.sol contract is responsible for managing and approving different types of users on the platform, such as system accounts, public minters, and creators. These accounts are held in mappings to provide access control to the DigitalMediaToken.sol. They can be modified over setter functions. As the public minters and the system accounts can be managed by the contract owner, the approved creators can be managed by the system accounts. The contract also keeps track of hashes already used for new digital media tokens.

DigitalMediaManager.sol

The abstract DigitalMediaManager.sol contract keeps track of the mapping between the digital media id and its digital media. It offers internal functions to create and burn it. Creating a new one increases a counter value to ensure each gets its unique id. Burning a digital media means setting the release count to its total supply such that no new tokens can be minted.

ERC721Token.sol

The abstract ERC721Token.sol contract inherits the ERC721.

The ERC721Burnable, and the ERC721Pausable implementations from the upgradeable Openzeppelin contracts. It allows the contract owner to change the baseUri and exposes a

public view function to retrieve the baseUri. It also allows the owner to pause and unpause the contract.

DigitalMediaReleaseManager.sol

The abstract DigitalMediaReleaseManager.sol contract combines the digital media with the ERC721 token. It contains internal functions to create and burn digital media releases.They are ERC721 tokens that are mapped to a corresponding digital media.

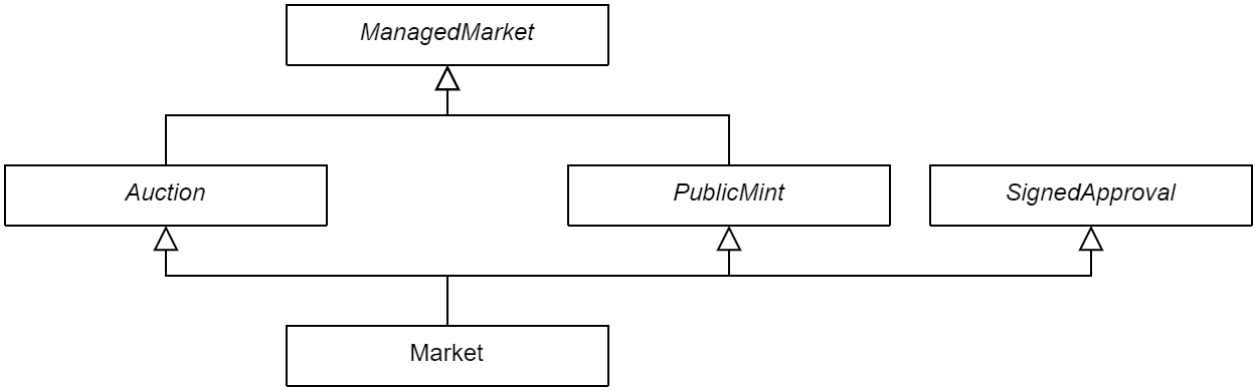
The create function for digital media releases first checks that the total supply will not be exceeded and then mints the requested count of ERC721 tokens. Then, the token is minted, and the corresponding digital media id and the release index are stored for the specific token. Finally, the release count is increased by the number of tokens minted.

Market.sol

The Market.sol contract is the marketplace for trading the DigitalMediaToken.sol. It allows the purchase of tokens using a signed message from a seller, including the buyer's address, the token id, the price, an approval id, and a deadline.

While purchasing, the signature is validated such that the token owner signed it; the approval is invalidated and then traded. The trade first sends commissions to the sale and the creator entity if they are set. Then the rest of the paid amount is sent to the seller.

Finally, the ERC721 token is sent to the buyer. The contract also allows the token owner to gift a token to another user directly. In addition, the digital media creator can directly sell a pre-minted token for a fixed price.



The Market, similar to the Token contract, distributes functionality over a set of abstract contracts. The ManagedMarket.sol cares about the parametrization of the market, the Auction.sol provides an English auction for token sales, the PublicMint.sol cares about the private or public minting of the tokens, and the SignedApproval.sol validates signatures.

ManagedMarket.sol

The abstract ManagedMarket.sol contract handles all configurations for the market. It holds properties for commissions like the sale, auction, and creator commission. In addition, the contract sets auction parameters like the minimum start auction price, the minimum increment of the price, or the minimal and maximal auction time. It allows the market to switch between enabled and disabled and exposes setter functions for all the values the contract owner can call.

PublicMint.sol

The abstract PublicMint.sol contract extension allows the collection creator to start a public or private mint for his collection. The creator can initialize such a public mint by setting parameters like if it is public, the price, or a whitelister address in case of a private mint. If it is a public mint, users can mint new tokens by providing the right ether amount, equal to the price, the media id, and the number of tokens to be minted.

If all checks are passed, a new ERC721 token is minted, the sale commission subtracted, and the rest of the amount sent to the seller. If the sale is private, the user can only buy the token if he can provide a valid signature containing his address generated by the whitelister. The contract also exposes a view function where a user can check if he has a valid permit to mint tokens and how many are still available.

Auction.sol

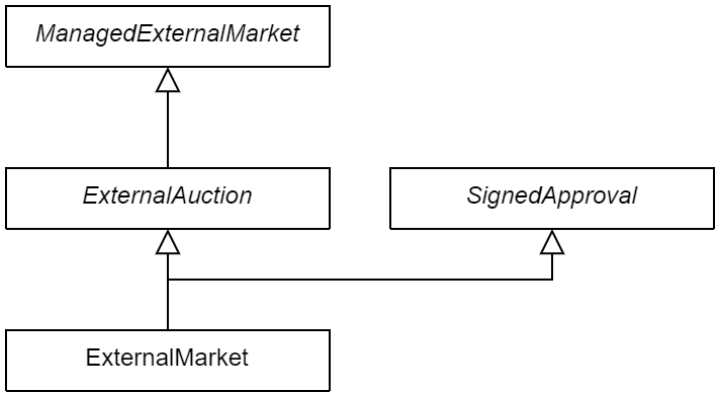
The abstract Auction.sol contract allows a token holder to start an English auction for it. By doing so, the token is transferred to the Auction.sol contract and an auction is started using an auction time and a start price. As long as the auction is open, anyone can make a bid by sending the ether value of his bid.

The bid value needs to be larger than the minimum increase of the bid. If the bidder is the new frontrunner of the auction, the old frontrunner gets paid back for his bid. The first bidder starts the auction from where the auction time starts. If there is a bidding war, the auction time is extended by a predefined amount of minutes since the last bid.

When an auction is finished, everyone can resolve it. This sends the auction and the creator commission to the authorities, the rest to the seller, and the token to the highest bidder. A token seller can also cancel an auction whenever he wants. This will send back the bid amount to the frontrunner and the token back to the seller.

ExternalMarket.sol

The ExternalMarket.sol contract is similar to the Market.sol but allows the trading of arbitrary, external ERC721 tokens. Except for the direct sale and public mint, the functionality of the external market overlaps the market contract.



ManagedExternalMarket.sol

The abstract ManagedExternalMarket.sol contract cares about similar properties as in the ManagedMarket.sol. The only difference lies in the setting of allowed collections and the royalty data to the external token owners. It can be set that only allowed collections can be used, and if so, the allowed collections can be set by the contract owner. If the external collection is Ownable or uses AccessControl, commissions (royalties) can be set to the owner or default admin of the external contract.

ExternalAuction.sol

The abstract ExternalAuction.sol uses the same functionality as the Auction.sol contract but for external ERC721 contracts.

SignedApproval.sol

The abstract SignedApproval.sol contract implements an internal helper function that can validate a signature by its parameter hash and signature components. It recovers the signer address from the parameters, checks if the signature was already used, and returns the signature address.

Best Practices

The Best Practices analysis does not cover direct vulnerabilities. It shows the overall project and code structure and if best practices were applied. This is a good measurement to validate the audit result, as a clean and understandable code base indicates that most bugs and vulnerabilities were found.

☑	The code was provided in a source control.
☑	A technical documentation was provided (Yes, inline NatSpec documentation).
☑	Minimal code duplication.
☑	Smart Contracts are unflattened.
☑	A recent solidity version was used (0.8.9).
☑	A framework for testing and deployment was used (Hardhat).
☑	There are tests.
☑	Tests are easy to run.
☑	There is no unused code.
☑	The code follows standard Solidity naming conventions.

The developers applied all best practices in the following code base. The code base contains unit tests for 100% line coverage, and the contracts and functions are well-commented.

Findings

The findings were categorized into the following four different levels:

- Critical: Potential loss of funds is expected. They need to be fixed immediately.
- Medium: Errors that can cause the contracts to fail. Manual changing needs to be done to restore the contract functionality.
- Low: Errors that can cause the contracts to fail in specific conditions like edge cases.
- Informational: Suggested improvements of the contracts that do not have security-related issues (e.g. gas optimization).

Critical

No Critical issues were found

Medium

No Medium issues were found

Low

WB1: Wrong bounded percentage values

In `contracts/ExternalMarket/ManagedExternalMarket.sol` and `contracts/Market/ManagedMarket.sol` the setter functions for the commission have a wrong maximum bound. It is assumed that each commission can not exceed 100%. However, they must not exceed 100% summed up. The administrator can therefore set the sum of the commissions larger than 100%, which will break the functionality of the contract, e.g. the auction.

The following functions are affected:

`contracts/ExternalMarket/ManagedExternalMarket.sol: setSaleCommission, setMaxCreatorCommission, setAuctionCommission`

`contracts/Market/ManagedMarket.sol: setSaleCommission, setCreatorCommission, setAuctionCommission`

Recommendation

Check that the sum of all the commissions does not exceed 100% when trying to set a new commission amount. For sale commission, for example, use: `require(value <= MAX_PERCENTAGE - creatorCommission - auctionCommission, “Value too large”)`

Status

Resolved in commit: 46b82559f8c36bf7b77bd4884fa66de837d35bd4

BC1: Bounded check missing

In `contracts/ExternalMarket/ManagedExternalMarket.sol` and `contracts/Market/ManagedMarket.sol` the `setMinAuctionTimeMinutes` and `setMaxAuctionTimeMinutes` are not checked to be larger or smaller than the other value.

When the max auction time is smaller than the min auction time, no new auction can be started.

Recommendation

Check the values before storing.

For `setMinAuctionTimeMinutes` use `require(value <= maxAuctionTimeMinutes, "Min auction time too long")`

For `setMaxAuctionTimeMinutes` use `require(value >= minAuctionTimeMinutes, "Max auction time too short")`

Status

Resolved in commit: 56fc8d0a295ab94cdc708f81908b359d5a5fa472

IC1: Initializer call missing

In `contracts/ExternalMarket/ExternalMarket.sol` and `contracts/Market/Market.sol` initializers for the reentrancy guard are missing. It is best practice to initialize all inherited initializers even if they do not implement essential logic.

Recommendation

Add `__ReentrancyGuard_init();` to the `initialize()` function of the mentioned contracts.

Status

Resolved in commit: 9c3cb19bc0731b3f5407a9b398d880058830150b

Informational

EF1: Set public functions to external

Functions that are only called from external sources should be declared external to optimize gas costs. The following functions are visible as public but are not accessed within the contract:

`contracts/Token/ManagedToken.sol: setApprovedPublicMinter, setApprovedCreator, setUsedMetadataHash`

`contracts/Token/ERC721Token.sol: setBaseUri, getBaseUri, setPaused`

`contracts/Token/DigitalMediaReleaseManager.sol: getTokenCreator`

`contracts/Token/DigitalMediaToken.sol: createDigitalMedia, createDigitalMediaAndReleases, createDigitalMediaReleases, updateCollection, burnDigitalMedia`

`contracts/ExternalMarket/ManagedExternalMarket.sol: setAllowedCollectionsOnly, setAllowedCollection, setContractEnabled, setCommissionCollectorAddress, setSaleCommission, setMaxCreatorCommission, setAuctionCommission, setBidIncrementPercentage, setBidTimeExtensionMinutes, setMinAuctionTimeMinutes, setMaxAuctionTimeMinutes, setMinAuctionStartPrice, setCollectionRoyalties`

`contracts/ExternalMarket/ExternalAuction.sol: auctionToken, bid, resolve, cancelAuction`

`contracts/ExternalMarket/ExternalMarket.sol: purchase, gift, invalidateApproval`

`contracts/Market/ManagedMarket.sol: setContractEnabled, setCommissionCollectorAddress, setSaleCommission, setCreatorCommission, setAuctionCommission, setBidIncrementPercentage, setBidTimeExtensionMinutes, setMinAuctionTimeMinutes, setMaxAuctionTimeMinutes, setMinAuctionStartPrice`

`contracts/Market/PublicMint.sol: userMintPublic, userMintPrivate, userMintAddressStatus`

`contracts/Market/Auction.sol: auctionToken, bid, resolve, cancelAuction`

`contracts/Market/Market.sol: purchase, gift, invalidateApproval, setDirectPurchasePrice, directPurchase`

Recommendation

Change the functions from `public` to `external`

Status

Resolved in commit: 047f62055b0b8e705cbb450ff550f93f185f4b39

UC1: Unnecessary check

In `contracts/ExternalMarket/ManagedExternalMarket.sol` on line `99` is an unnecessary check. If the first condition is false (in this case, `allowedCollectionsOnly` is true), the `allowedCollectionsOnly` check to be true is not necessary.

Recommendation

Change the require statement to:

```
require(!allowedCollectionsOnly || allowedCollection[collection], "Collection not in allowlist");
```

Status

Resolved in commit: 0db3c1e493b16d8312b6c184ef95ffca81ed196e

WD1: Wrong description

In `contracts/Token/ManagedToken.sol`, the description about the contract is wrong (description from signed approval).

Recommendation

Change the description to the contract functionality.

Status

Resolved in commit: 180bb1e2b02fa2627d2db91f13274abb9c1ba53a

Limitation

This code review was conducted carefully and on a best-effort basis. However, this does not guarantee that there are any undiscovered issues and vulnerabilities. This audit gives no warranties on the security of the code.