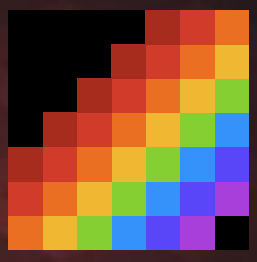
Blaize.Security



DIGITAL ORIGINAL

DIGITAL ORIGINAL
SMART CONTRACT AUDIT

June 28th, 2024 / V. 1.0

Table of Contents

Executive Summary

During the audit, we examined the security of the Auction smart contract for the DigitalOriginal protocol. Our task was to find and describe any security issues in the platform's smart contract. This report presents the findings of the security audit of the DigitalOriginal platform conducted between March 22, 2024, and June 20, 2024.

During the audit, the Blaize Security team audited the Auction and ArtToken contracts together with several utility contracts. The ArtToken is a regular ERC721 contract enhanced with the purchase functionality. The Auction contracts implement the regular plain auction logic for each token within the collection. More information can be found in the protocol overview section. Therefore, the auditors' goal was to analyze the security of funds, the correctness of the auction mechanism, and the correctness of the ERC721 standard implementation - these and other potentially vulnerable areas (including several internal checklists).
During the audit, the auditors identified one high, two low, and a few information issues. Issues were connected to clarifying the support of purchase via smart contracts, issues with the frontrun of the purchase, corrections of the auction flow (to avoid duplicated auctions), parameter validations, and best practices support. The DigitalOriginal team promptly addressed all these issues, either fixing or verifying them.

Also, auditors should note several points:

1. NFT is upgradeable - thus, the owner can change any aspect of its logic. While upgradeability is a regular practice, it creates a controllable backdoor with a risk of exploitation in case of inappropriate private key handling. Usually the ERC721 smart contract is implemented as non-upgradable since it represents funds stored on the user's balance. Thus, it creates centralization and possible issues in case of the leak of the admin's private key; some marketplaces or exchanges may not list tokens with upgradable implementation. It is a negative factor for security as it violates the check against backdoors in the code.

2. ArtToken transfer restriction (receiver should not be a contract) is not safe from the override completely. The check against the code length does not work for contracts under construction or pre-calculated addresses. Therefore, it is possible to bypass the check if the contract performs the operation within its constructor. This case cannot be covered in EVM chains due to this flow. This is a general warning that such a case is possible. The team verified that they are aware that such an approach eliminates multisigs and smart wallets as well.
3. Single point of failure risk. The ArtToken contract currently depends on the ADMIN (set as an immutable variable). The ADMIN is responsible for correct signatures during the auction of NFTs. However, as the variable is immutable, in the case of the compromised ADMIN, the NFT will be exploitable, with all further issued signatures issued from the ADMIN treated as invalid. And the only procedure to change the exploited ADMIN is to upgrade the NFT implementation - in case if ADMIN and deployer (proxy owner) represent different entities. Therefore, the team should provide a high level of security for the ADMIN private key, including the usage of multisigs and/or cold-wallet approaches. As for now, the point violates the check against the single point of failure.
4. Validation of the correct receivers of the NFT price shares is on the operator's (admin) side and is delegated to be out of the contract. Therefore, the DigitalOriginal team should verify the existence of the sanitizing actions regarding the recipients' validation.

Also, auditors noted the high quality of the code, good natspec comments (though the general documentation is absent), high coverage with native unit tests, and good organization of the project and development pipeline. From all points except the noted risks, the total security of the smart contract is high, so the project successfully passes the audit.

Auditing strategy and Techniques applied/Procedure

Blaize.Security auditors start the audit by developing an auditing strategy - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

MANUAL AUDIT STAGE

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control / roles structure review, analysis of user and admin behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security one;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

TESTING STAGE:

- Development of edge cases from manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

Audit Rating

Score: 9.8 /10

	RATING
Security	9.8
Logic optimization	9.9
Code quality	10
Testing suite	10
Documentation	9

Security: General mark for the security of the protocol.
The main mark for the audit qualification.

Logic optimization: Evaluation of how optimal the implementation is, including presence of extra/unused code, uncovered/extra cases, gas (or its analog) optimization, memory management optimization, etc
Code quality: Evaluation of best practices followed, code readability, structure and convenience of further development
Testing suite: Availability of the native tests suite, level of logic coverage, checks of critical areas being covered.
Documentation: Availability and quality of the documentation, coverage of core functionality and user flows: whitepaper, gitbook, readme, specs, natspec, comments in the code and other possible forms of documentation.

SECURITY RATING CALCULATION

Approximate weight of unresolved issues.
Critical: -3 points
High: -2 points
Medium: -0.5 points
Low: -0.1 points
Informational: -0.1 point (in general, depends on the context)

Note: additional concerns, violated checklist items (including standard vulnerabilities), and verified backdoors may influence the final mark and weight of certain issues.

Starting with a perfect score of 10:

Critical issues: 0 issue (0 resolved): 0 points deducted

High issues: 1 issue (1 resolved): 0 points deducted

Medium issues: 0 issue (0 resolved): 0 points deducted

Low issues: 2 issues (2 resoled): 0 points deducted

Informational issues: 3 issues (1 resolved, 2 verified): 0 points deducted

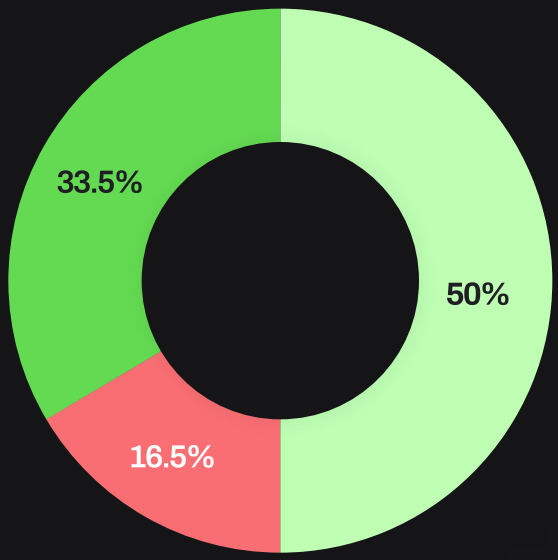
-0.1 points deducted regarding the risk of single point of failure
-0.1 points deducted for the controllable backdoor in a form of the upgradeable contract

Security rating = 10 - 0.2 = 9.8

Technical Summary

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

- Critical
- High
- Medium
- Low
- Info



The table below shows the number of the detected issues and their severity. A total of 6 problems were found. 6 issues were fixed or verified by the Customer's team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	1	1
Medium	0	0
Low	2	2
Info	3	3

SEVERITY DEFINITION

CRITICAL

The system contains several issues ranked as very seriousand dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

HIGH

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

MEDIUM

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

LOW

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

INFO

The issue has no impact on the contract's ability to operate, yet is relevant for best practices. Or this status can be assigned to the issues related to the suspicious activity or substandard business logic decisions which cannot be classified without the comments from the team (and can be re-classified on the later audit stages).

Issues reviewed by the team can get the next statuses:
Resolved: issue is resolved by an appropriate patch or changes in the business logic
Verified: the team provided sufficient evidences that the issue describes desired behavior
Unresolved: neither path nor comments provided by the team, or they are not sufficient to resolve the issue
Acknowledged: the team accepts the misbehavior and connected risks

Audit Scope

Language/Technology: **Solidity**

Blockchain: **Ethereum**

The scope of the project includes:

- contracts\art-token\ArtTokenBase.sol
- contracts\art-token\ArtToken.sol
- contracts\auction-house\AuctionHouseStorage.sol
- contracts\auction-house\AuctionHouse.sol
- contracts\utils\Deployer.sol
- contracts\utils\Distribution.sol
- contracts\utils\EIP712.sol

Repository: https://github.com/digital-original/contracts

The source code of the smart contract was taken from the branch: **master**.
Initial commit:

- 67d68b95104b1877183c6a741679e6c7c85438c1

During the audit process, code fixes occurred in the respective PRs:

- https://github.com/digital-original/contracts/pull/27
  https://github.com/digital-original/contracts/pull/28
  https://github.com/digital-original/contracts/pull/29
  https://github.com/digital-original/contracts/pull/30
  https://github.com/digital-original/contracts/pull/31

Fixes were lately merged into the branch: **master**
Final commit:

- 47864e04acbebd242b7e8928c98e25766f4f469a

Protocol overview

DESCRIPTION

The **ArtTokenBase** contract provides an upgradeable implementation of the ERC721 standard, incorporating enumerable and URI storage extensions from the OpenZeppelin library.
The **ArtToken** contract extends **ArtTokenBase**, adding functionalities specific to creating, purchasing, and managing Digital Original NFTs. It ensures that only **AuctionHouse** can mint tokens.
The **AuctionHouseStorage** is a library that manages the storage layout, ensuring the auctions' data is securely stored.
The **AuctionHouse** contract implements the core auction functionalities for Digital Original NFTs and provides payment handling. It leverages **AuctionHouseStorage** to manage its data.
ArtToken and **AuctionHouse** contracts also integrate EIP712 for typed structured data hashing and signing, ensuring verifiable transactions.
The Deployer contract facilitates the deployment of upgradeable contracts, specifically the **ArtToken** and **AuctionHouse** contracts, using the Transparent Proxy pattern.
The Distribution library provides functionality for distributing ERC20 token rewards among participants based on predefined shares and ensures the validity of distribution.
The EIP712 contract provides functionality for verifying signatures according to the EIP712 standard.

ROLES AND RESPONSIBILITIES

ArtToken

1. Admin: Admin address is only used internally for validating signatures during token purchases.
2. Platform: Accepts protocol fees.
3. AuctionHouse: Mints new NFTs.
4. Regular user:

- Buys tokens.
- Grants necessary allowance of USDC.
- Interacts with regular ERC721 functionality

AuctionHouse

1. Admin: Admin address is only used internally for validating signatures during the auction creation.
2. Platform: Accepts protocol fees.
3. Regular user:

- Creates an auction (with the appropriate signature from the Admin), raises the price and finishes the auction.
- Provides necessary allowance of USDC.

LIST OF VALUABLE ASSETS

ERC721 Tokens (ArtToken on the AuctionHouse contract)

- Minted ERC721 Tokens: These tokens represent Digital Original NFTs, which are created, tracked, and transferred within the ArtToken contract. These tokens can be placed at the auction and minted using the AuctionHouse contract.

USDC (ArtToken, AuctionHouse contract)

- USDC Tokens: USDC bids are collected and stored temporarily within the AuctionHouse contract until the auction ends.

Note: The protocol works with the exact balance of the USDC transferred into/out of the contract; therefore, contracts do not have refund functionality for funds sent by mistake.

SETTINGS

ArtToken.sol
- The ADMIN address is set during deployment and is required for a valid signature while purchasing NFTs.
- PLATFORM is set during deployment and receives a fee from NFTs purchase.
- AUCTION_HOUSE address is set during deployment and is responsible for minting tokens within the ArtToken contract.
- The USDC is set during deployment and is represents the accepted stablecoin.

AuctionHouse.sol
- The ADMIN address is set during deployment and provides a valid signature while creating an auction.
- The PLATFORM address is set during deployment and receives a fee after finishing the auction.
- the TOKEN address is set during deployment. It is the main asset auction is interacting with.
- USDC address is set during deployment, responsible for payments within contract.

DEPLOYMENT

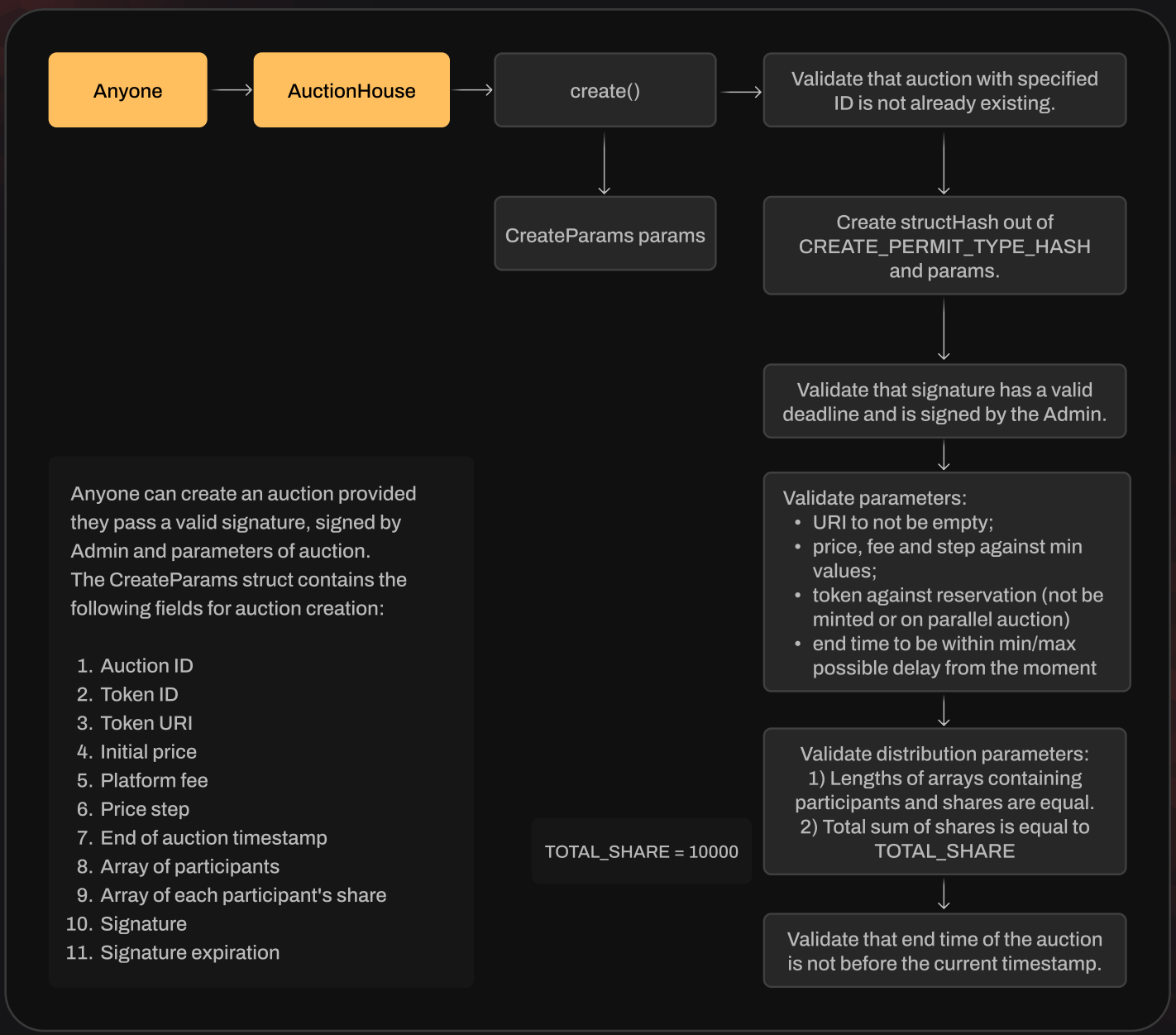
Standard Hardhat deployment scripts are prepared for contract deployment. The `deploy-classic` is a helper script to deploy any contract with passed parameters. The `deploy-contracts` script functionality manages the deployment of the Deployer contract and the related ArtToken and AuctionHouse contracts. It sets up the proxy and admin configurations, retrieves necessary contract instances, and returns deployment information.

`deploy-upgrade` and `deploy-upgradeable` scripts are designed to handle the deployment and upgrade process for smart contracts using the proxy pattern, ensuring that contracts can be upgraded without changing their address, thereby preserving state and functionality.

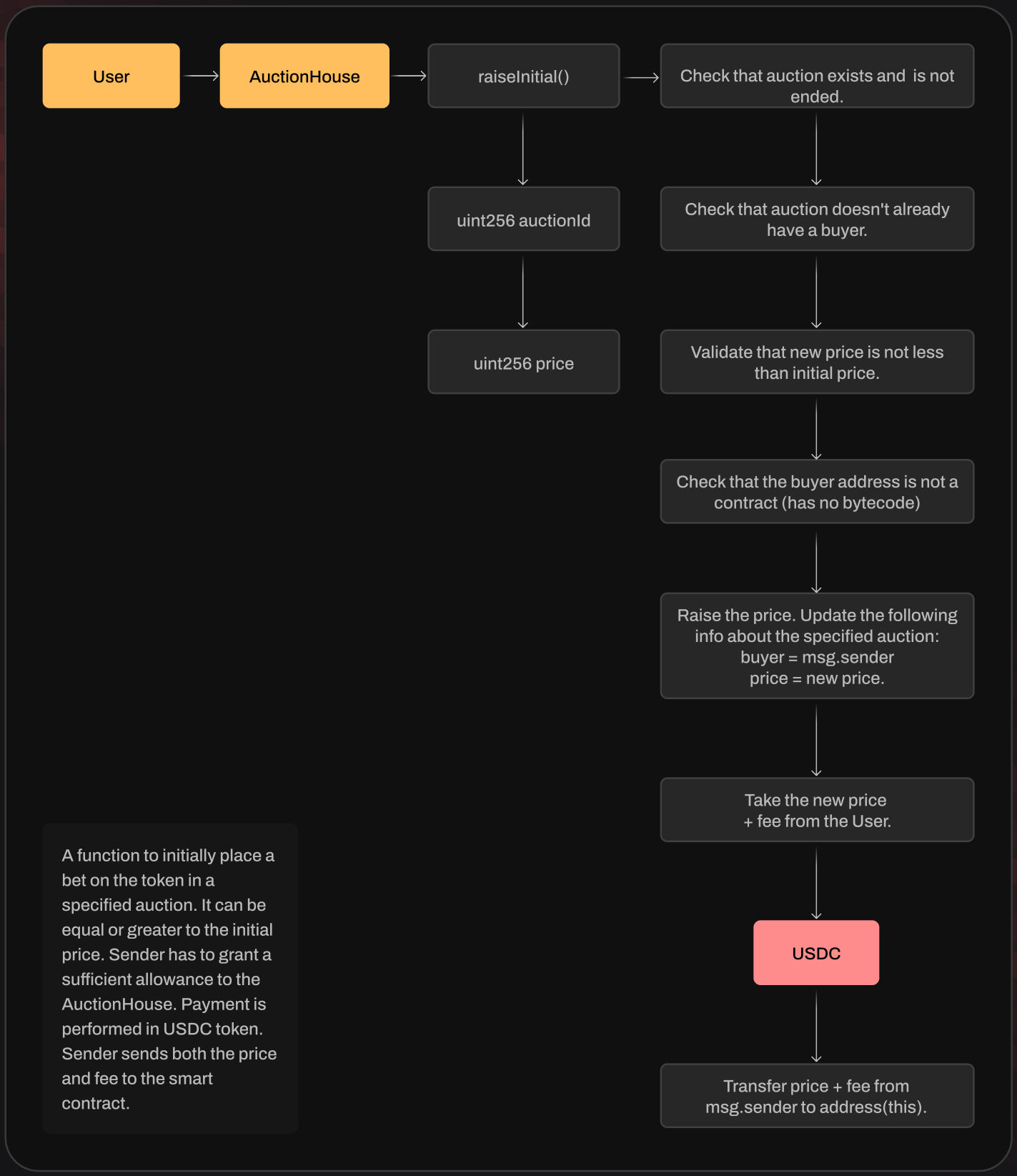
Note! Deployment is performed via Deployer.sol, which deploys ArtToken and AuctionHouse within the constructor.

DEPLOYMENT FLOW

CREATE AUCTION FLOW

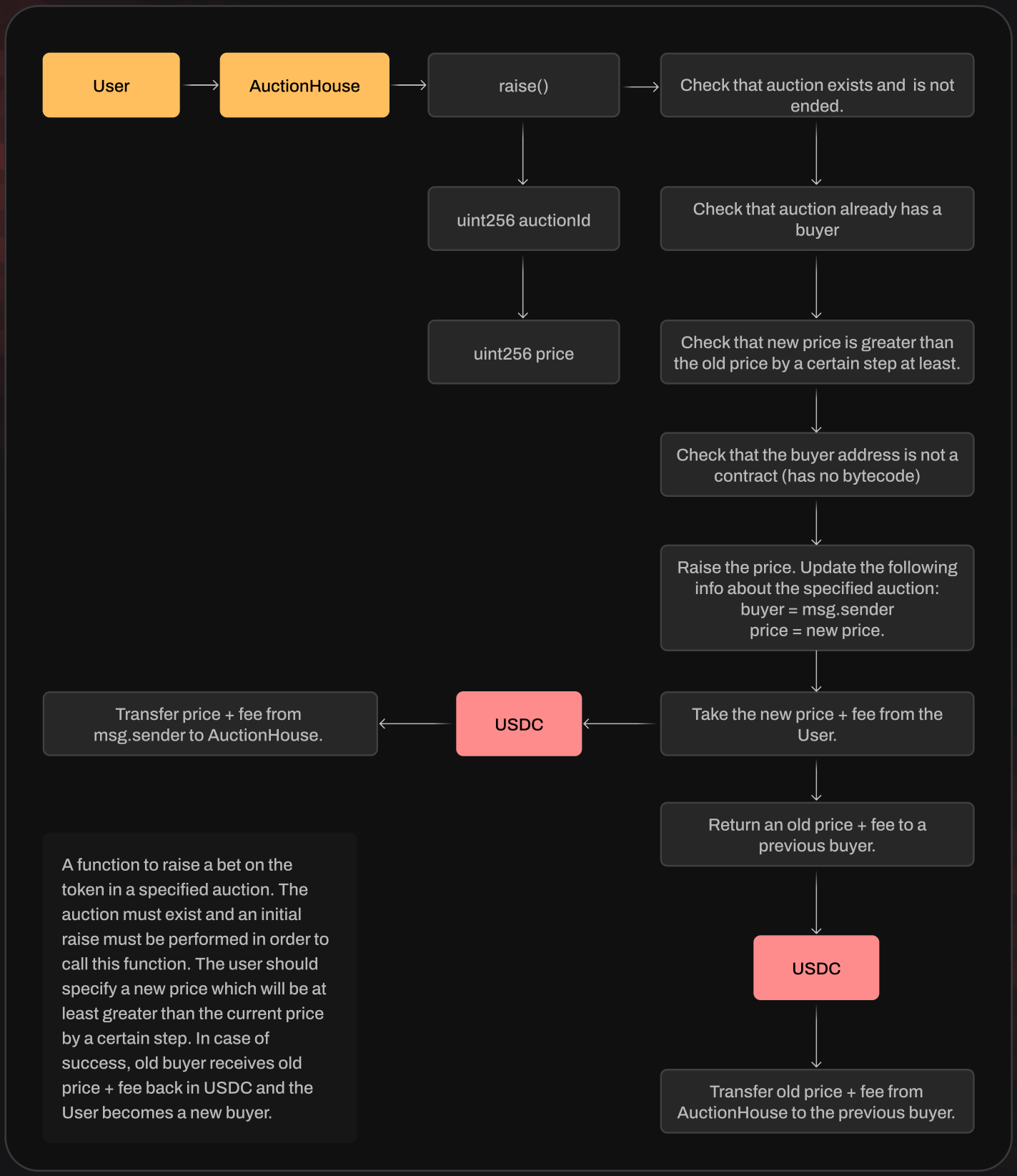


INITIAL RAISE FLOW



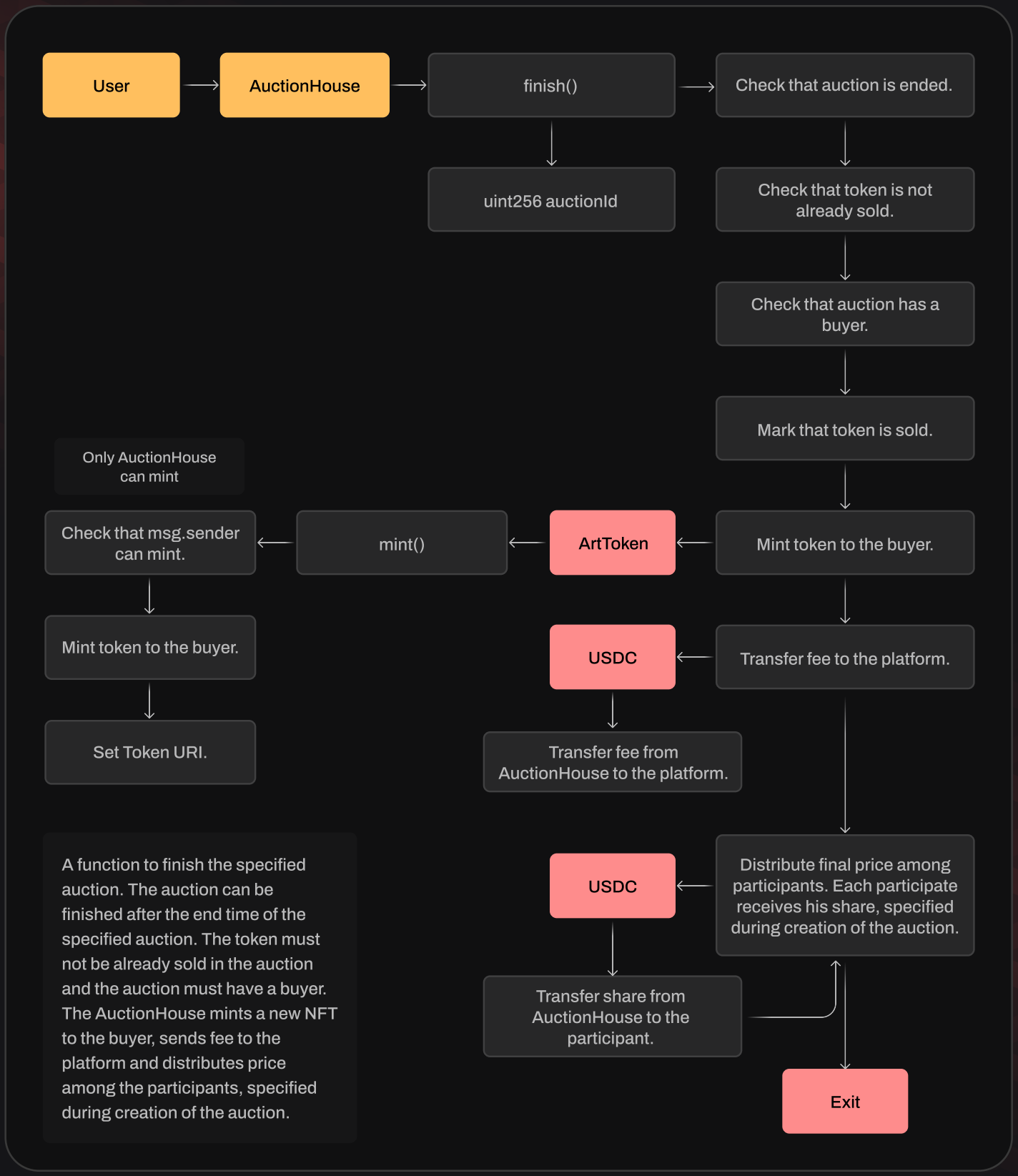
A function to initially place a bet on the token in a specified auction. It can be equal or greater to the initial price. Sender has to grant a sufficient allowance to the AuctionHouse. Payment is performed in USDC token. Sender sends both the price and fee to the smart contract.

RAISE FLOW

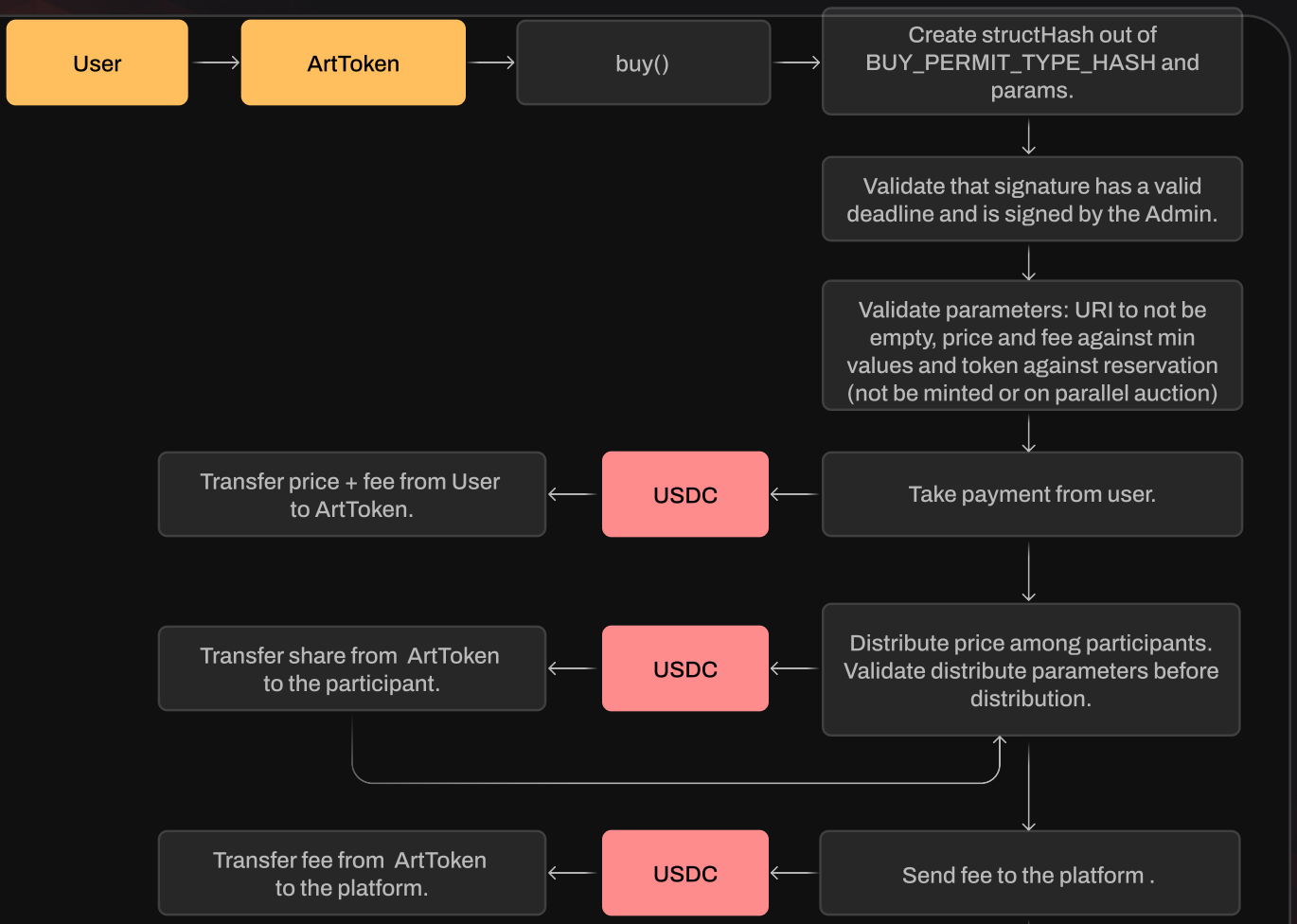


A function to raise a bet on the token in a specified auction. The auction must exist and an initial raise must be performed in order to call this function. The user should specify a new price which will be at least greater than the current price by a certain step. In case of success, old buyer receives old price + fee back in USDC and the User becomes a new buyer.

FINISH AUCTION FLOW



BUY TOKEN FLOW



Allows a user to purchase a token provided he passes a valid signature signed by the Admin which allows the user to buy token an parameters. The parameters struct consists of:

1. Token ID
2. Token URI
3. Price
4. Platform fee
5. Array of participants
6. Array of each participant's share
7. Signature
8. Signature expiration

The user pays for the Token in USDC.
User must pay price + fee. The fee is sent to the platform while the price is distributed among the participants.

CONTRACTS' STORAGE

AuctionHouse

Immutable:
- Admin
- Platform
- Token
- Usdc
- min duration

Constants:
- max duration (21 days)
- min price (100 usdc)
- min fee (100 usdc)

Layout:
- auctions
- tracked token ids

Layout consist of different auctions.
Each auction contains the following information:
1. ID of the token
2. Price of the token
3. Platform fee
4. Price step
5. End of the auction
6. Buyer
7. Flag for indication if token is sold
8. Token URI
9. Array of participants for reward distribution
10. Share of each participant

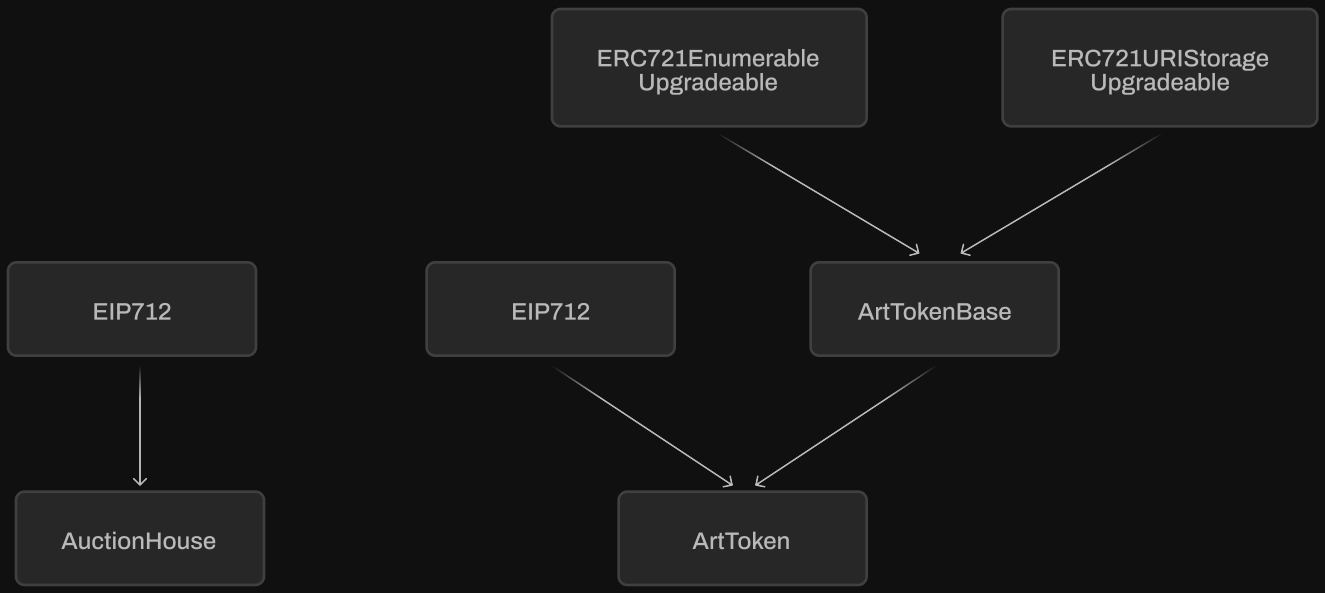
ArtToken

Immutable:
- Admin
- Platform
- Auction house address
- Usdc

Constants: min price and fee (100 usdc)

Depencencies:
ERC721EnumerableUpgradeable
ERC721URIStorageUpgradeable

INHERITANCE TREE



Complete Analysis

STANDARD CHECKLIST / VULNERABLE AREAS

Storage structure and data modification flow	Pass
Access control structure, roles existing in the system	Pass
Public interface and restrictions based on the roles system	Pass
General Denial Of Service (DOS)	Pass
Entropy Illusion (Lack of Randomness)	N/A
Order-dependency and time-dependency of operations	Pass
Accuracy loss, incorrect math/formulas other violated operations with numbers	Pass
Validation of function parameters, inputs validation	Pass
Asset management, funds flow and assets conversions	Pass
Signatures replay and multisig schemes security	Pass
Asset Security (backdoors connected to underlying assets)	Fail
Incorrect minting, initial supply or other conditions for assets issuance	Pass
Global settings mis-using, incorrect default values	Pass
Violated communication between components/modules, broken co-dependencies	N/A
3rd party dependencies, used libraries and packages structure	Pass
Single point of failure	Fail
Centralization risk	Pass
General code structure checks and correspondence to best practices	Pass
Language-specific checks	Pass

HIGH-1 Resolved

FRONTRUN POSSIBLE WHEN BUYING A TOKEN

ArtToken.sol: buy().
The function is available for anyone to call if the correct signature is provided (from the Admin). The signature contains several parameters necessary for the buying. If the signature is valid, the user pays USDC to mint an ERC-721 token to his balance. However, the signature doesn't include the address of the msg.sender (of the actual caller). Thus, when a user sends a transaction with the signature and it appears in the mempool, an attacker is able to see the signature, send the transaction with it, and mint the token in his balance instead of the initial user.

The issue is marked as High since the attacker still has to pay for the token. However, if the signature is supposed to be personal and used by a certain person, the attacker can steal it. The issue also requires validation and may be decreased to informational in case the signature is supposed to be public, and the first user who uses it can buy a token.

RECOMMENDATION:

Include msg.sender in signature OR provide clear documentation and notice in the dApp interface that anyone can use the signature on the FIFO rule for the buy() function call.

POST-AUDIT:

msg.sender address was added to the signature
https://github.com/digital-original/contracts/pull/28

LOW-1 Resolved

MISSING BOUNDARIES VALIDATION FOR THE PARAMETER

AuctionHouse.sol: create().

- params.price, params.fee, params.step - require checks for boundaries (min/max)
- params.deadline - require a check for the max boundary

It is recommended to have several validations for each parameter:

- secure default value
- min and max boundaries
- extreme values (in case boundaries are not introduced)

Though no default value is set (including deployment/tuning/upgrade scripts), and there are no min/max values to validate against, the parameter is exposed to the risk of misconfiguration and/or human error.

The issue is marked as Low, as the authorized role controls the parameters despite the risk of misconfiguration. However, changing the incorrectly set parameter may take some time, thus creating a window of instability.

RECOMMENDATION:

Provide the default value for the parameter, introduce reasonable min/max boundaries for the parameter.

POST-AUDIT:

The DigitalOriginal team added validation for the minimal possible price and fee, both as constants for 100 USDC. The team verified that the maximum boundary cannot be set by the protocol's design, where each NFT may represent objects of different scales (in terms of price). The same applies to the step - thus, it is validated not to be 0. Also, the security team verified the deadline validation, which was performed using the appropriate backend service. The deadline for the signature is set in boundaries from 1 to 10 minutes.

LOW-2 Resolved

AUCTION CAN BE STARTED FOR THE ALREADY MINTED NFT

AuctionHouse.sol: create()
The contract validates the auctionID to be unique; however, it does not check that the tokenID is already minted. Thus, the contract allows creating an auction for already sold NFT—or even for the NFT that is currently on sale—thus confusing the community.

Since the auction creation requires ADMIN signature, the issue is marked as Low, though it may cause funds to be stuck on the contract in case a duplicate auction is created.

RECOMMENDATION:

Add the verification for the token not to be minted.

POST-AUDIT:

The code was updated to include a token reservation - now auction cannot be created for the minted token (token with owner) and currently auctioned token cannot be minted or added to another auction.
https://github.com/digital-original/contracts/pull/27

INFO-1 Verified

INITIAL PRICE RAISE DOES NOT REQUIRE A STEP

AuctionHouse.sol: raiseInitial()
Although all next raises require a step to change the price on the auction, the initial raise from the creation price works without it. While the issue creates no security risk, it looks either as sub-standard behavior or as a missed case. Therefore, additional verification from the team is required.

RECOMMENDATION:

Verify that the initial price raise does not require a step.

POST-AUDIT:

The DigitalOriginal team verified that the initial price is not dependent on the step, as it should match or exceed the initial price in the free range.

INFO-2 Resolved

_SAFEMINT IS NOT USED

ArtTokenBase.sol: _mintAndSetTokenURI().
_safeMint validates that if the token's receiver is a smart contract, it must implement the checkOnERC721Received() hook. The issue was raised since both AuctionHouse and ArtToken allow other smart contracts to call functions for purchasing tokens.
The contract currently states that purchases during the sale are disallowed, though that will be changed later. Therefore, the security team still requires verification from the DigitalOriginal team for the post-sale activities. Though the issue is set as Info, it might lead to the tokens being stuck on smart contracts that don't implement a necessary interface.

RECOMMENDATION:

Verify that NFTs can be minted in later stages after the initial sale, and validate if other smart contracts should be able to receive tokens immediately after minting. Use _safeMint instead of _mint if other smart contracts should be able to buy tokens. Also validate that receiving smart contracts will implement checkOnERC721Received() (the ones which will interact with functions AuctionHouse: raise(), raiseInitial(), ArtToken: buy())

POST-AUDIT

The DigitalOriginal team verified that the protocol will not support smart contract buyers (including multisigs and other smart wallets). Therefore, in this case, usage of regular mint is more gas effective, and the absence of the check onERC721Received is verified. Also, the team added the restriction against contracts into the “raise” methods of the auction (in the same way as in the mint function) to restrict contracts from participation in the auction (and thus - cutting off edge cases which could lead to stuck funds).
https://github.com/digital-original/contracts/pull/30

INFO Resolved

BEST PRACTICES AND CODE STYLE VIOLATIONS, OPTIMIZATIONS

1) AuctionHouse.sol, ArtToken.sol: constructor().
All constructor parameters should be validated so they are not equal to zero addresses, especially since the storage variables are immutable and can't be changed later. Although the functions are restricted, validating each address against the address(0) is recommended to minimize the risk of human error and incorrect value set.
Note: As ADMIN is used as a signer, it is especially crucial to validate against address(0).
2) AuctionHouse.sol: create(), ArtToken.sol: buy().
params.tokenURI should be validated as not being an empty string.
3) EIP712.sol, _requireValidSignature()
validate, that signer is not address(0) to have the correct signature checking flow

RECOMMENDATION:

Consider correction of all listed issues to increase the code readability and logic optimization.

POST-AUDIT

https://github.com/digital-original/contracts/pull/29

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

ArtToken / AuctionHouse

Scenarios to cover missed cases

- ✓ If zero price and fee provided, the mint won't be completed
- ✓ Upgradeability test for secondary initialization
- ✓ Trying to mint from unauthorized account fails
- ✓ Transfer (mint) to smart-contract address fails
- ✓ Returns correct data for supported interfaces
- ✓ Operations fails if the auction does not exist

Additional scenarios and hypothesis

- ✓ Should allow to buy token and revert on create function on the same token id
- ✓ Should revert on creating a secondary auction for the same toke id - if the current auction is raised but NFT is not minted yet
- ✓ Should revert on creating with existing token id until deadline
- ✓ Should allow to create auction for existing token id after deadline
- ✓ Should revert on creating auction with minted token id from previous auction
- ✓ Test on purchasing token via the contract in construction phase.

NATIVE TESTS OVERVIEW

The DigitalOriginal team shared the set of native tests - all of them were checked during the testing stage for completeness and integrity. Set of tests has high quality and the coverage sufficient by the security standard. Therefore, Blaize Security team focused on testing hypothesis and manual exploratory testing of the code.

Disclaimer

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.