Cyberscope

Audit Report

DeFi Kingdoms

March 2023

AuctionHouseAdminFacet	2e6f3165b1d323c306a45b5ac6e8178e788bd1d4ab45742b2f5503851ea7f820
AuctionHouseBase	1a299640a9a70c1aee299799390ad8ea28a3d67ad4f43151578af718e35308af
AuctionHouseGettersFacet	a51855fa5913698d56d69d41a12517399c3f8f2d796c2b43c99dcaf5acc8c2fc
AuctionHouseInit	6d7aacdd03aaee3d7048c6892a6166e44bf5200a083873f7df6becd5be2c1dc0
AuctionHouseMainFacet	69229e94e4a9100f16929355d51779fda990e01bf5c674fba1c4029e7b0408e4

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	14 Mar 2023

Source Files

Filename	SHA256
AuctionHouseAdminFacet.sol	2e6f3165b1d323c306a45b5ac6e8178e788bd1d4ab45742b2f5503851ea7f820
AuctionHouseBase.sol	1a299640a9a70c1aee299799390ad8ea28a3d67ad4f43151578af718e35308af
AuctionHouseGettersFacet.sol	a51855fa5913698d56d69d41a12517399c3f8f2d796c2b43c99dcaf5acc8c2fc
AuctionHouseInit.sol	6d7aacdd03aaee3d7048c6892a6166e44bf5200a083873f7df6becd5be2c1dc0
AuctionHouseMainFacet.sol	69229e94e4a9100f16929355d51779fda990e01bf5c674fba1c4029e7b0408e4
diamond-facets/helpers/token_fees/ITokenFeesInternal.sol	9775e9c5e1b45ef3e5e9fdbd9e45bddfe560642e064f078024a80fa13191204c
diamond-facets/helpers/token_fees/TokenFees.sol	d1489746ce7e7376eb72254c795ed5bf9885195e998cd3137184d24801e236a7
diamond-facets/helpers/token_fees/TokenFeesInternal.sol	0faf3d27124aea3143b582d3797791001d8cc8746e9735aad6b7ee35a2b9a32b
diamond-facets/helpers/token_fees/TokenFeesStorage.sol	fa80b8ee569c9fdf9be70846a6fd10396d3f47ed6f015814da50e3694bfcb5fa
diamond-facets/token/ERC1155/receiver/ERC1155Receiver.sol	6d04d70c81d9472df247197f6817022f24294201993f12e8a73bb401eeb8f619

diamond/Diamond.sol	f13e6562a47d495163812d04e27e3e737554360695f0063840d74a9cf5e5b5cc
diamond/DiamondUpgradeable.sol	849824c9fe67dcc36e9bf50fbbd23f25fe64d97b0802f3b55d1b622e17fd79b1
diamond/facets/DiamondCutFacet.sol	b718c1c71860e66752ba83210ae166a89b829fa806a2237cbcd1cd0b1d1b529b
diamond/facets/DiamondLoupeFacet.sol	af036ee66c4398a029bf4567b6326704431418c333bbee0029e53039c3adcbd3
diamond/facets/OwnershipFacet.sol	6978f53de20634d7c62215e2a81fd76fe0dd200821726d8f062fc76cda7c23f9
diamond/facets/Test1Facet.sol	cdbb6ff0645a655afb96f03eb4de8475517958529fcedd3b4129775518c14bfd
diamond/facets/Test2Facet.sol	769fd2d488695a9e0115613d600dfb57211d0277030edfc97e2c19b17f4fdbd9
diamond/interfaces/IDiamondCut.sol	0baf446ea5e18892e656d597353c3805605c55d6ae4436a4f110f432fff065eb
diamond/interfaces/IDiamondLoupe.sol	e4132a0a73c1956d04d068af156ebb62d2aad055c286de66e09305f71222c0d1
diamond/interfaces/IERC165.sol	7b90d86cddb8cd071e7ecc55652fe5f5a5ed06fb814d67315653a63c51677866
diamond/interfaces/IERC173.sol	34a78a9d4ccca7953af455ff43cd4ea9962c19648c9437cbd4bcfc48d83d3393
diamond/libraries/LibDiamond.sol	abd0d926c2acb9dab263000f679b5ad9341f4c5fc766c651d09f6bc0322deba4
diamond/upgradeInitializers/DiamondInit.sol	3b52b045b45fe110403702e1d1fe79c96c5e9723855650e8d875707c105c0c48
helpers/IPowerToken.sol	5c2ed4c0d414f49fbed73b469c542f28fbbbaa0f07bbd960c6cf6cc6285c4258
helpers/ITokenUnlocker.sol	473c72c6b432410ac86b94151b575188afdda0e853403f11af0c328a4c415638

LinkedListLibrary.sol	fa1bebdf8be34222f7ab8363bbbc7138de21df04c918e36f386f7c3b4299550f
OrderTypes.sol	4366b799c39a02f4ce81fb1f72b639eb9596b9647efc0c1d9ba3b48989e33004
RBTLibrary.sol	57c0c2021625c61595c025e1d9b50de522ae982d3a06face0d33828caa48a8c5
token_fees/ITokenFeesInternal.sol	9775e9c5e1b45ef3e5e9fdbd9e45bddfe560642e064f078024a80fa13191204c
token_fees/TokenFees.sol	d1489746ce7e7376eb72254c795ed5bf9885195e998cd3137184d24801e236a7
token_fees/TokenFeesInternal.sol	4c79a89eac6d9049fd54304a6cdc4d31a2c37fc9062226654d4a320aca6f82e7
token_fees/TokenFeesStorage.sol	9f32dffb41f640c004930ee0176a7c3327df5205d5d248c27e04274bde0c8d6e

Introduction

DeFi Kingdoms is an ecosystem of various smart contracts. This audit focuses on the following:

- `AuctionHouseBase.sol`: The base contract from which the rest inherit its functionality.
- `AuctionHouseAdminFacet.sol`: A contract that handles administrative actions like setting fees and pausing/unpausing the contract.
- `AuctionHouseInit.sol`: An initialization contract.
- `AuctionHouseGettersFacet.sol`: A contract that contains getter functions that return useful information or values.
- `AuctionHouseMainFacet.sol`: The contract that handles most of the heavy work of the auction, like adding and removing orders.
- `LinkedListLibrary.sol`: The LinkedListLibrary provides functionality for implementing data indexing using a circlular linked list. It was forked from https://github.com/o0ragman0o/LibCLL while adding a few lines of code.
- `OrderTypes.sol`: The OrderTypes acts as a place for keeping all kinds of data structures.
- `RBTLibrary.sol`: The RBTLibrary is a Solidity Red-Black Tree binary search library to store and access a sorted list of unsigned integer data. It was forked from https://github.com/bokkypoobah/BokkyPooBahsRedBlackTreeLibrary and only some error handling code was added to it.

which implement an auction house functionality for ERC20 and ERC1155 compliant tokens. The contracts allow users to submit buy and sell orders for any token against the same base power token. Orders that can be filled from the existing order book are executed immediately, and any unfilled quantities can be optionally added to the order book. The contracts are agnostic about how many decimals the paired tokens have, but it is recommended that dApps implement a UI that allows users to provide unit prices rather than total prices and to restrict quantities and unit prices to a safe range. The contracts also provide a formula for converting prices stored in the contract back into standard units. The contracts implement various roles and permissions, including a moderator role.

Roles

The contracts include two roles, the user and moderator role.

Moderator

The moderator role has authority over the following functions:

- `AuctionHouseAdminFacet`
  - `function pause()`
  - `function unpause()`
  - `function setDefaultFee(Side _side, uint256 _fee)`
  - `function setMarketFee(address _token, Side _side, uint256 _fee)`
  - `function setFees(address[] memory _feeAddresses, uint256[] memory _feePercents)`

User

The user can interact with the following functions:

- `AuctionHouseBase`
  - `function getBestOrder(address _token, Side _side)`
  - `function getBestOrder(address _token, uint256 _tokenId, Side _side)`
  - `function calcFee(address _token, Side _side, uint256 _quantity)`
- `AuctionHouseInit`
  - `function upgrade()`
  - `function init(address _powerTokenAddress, uint256 _nextOrderId)`
- `AuctionHouseGettersFacet`
  - `function getQuantityUpToPrice(address _token, Side _side, uint256 _price)`
  - `function getQuantityUpToPrice(address _token, uint256 _tokenId, Side _side, uint256 _price)`

- function getCostForQuantity(address _token, Side _side, uint256 _quantity)
- function getCostForQuantity(address _token, uint256 _tokenId, Side _side, uint256 _quantity)
- function getUserOpenOrders(address _user)
- function getPrices(address _token, Side _side)
- function getPrices(address _token, uint256 _tokenId, Side _side)
- function getOrdersAtPrice(address _token, Side _side, uint256 _price)
- function getOrdersAtPrice(address _token, uint256 _tokenId, Side _side, uint256 _price)
- function getNumPrices(address _token, Side _side)
- function getNumPrices(address _token, uint256 _tokenId, Side _side)

- AuctionHouseMainFacet
  - function makeSellOrderERC20(address token, uint256 powerTokenQuantity, uint256 tokenQuantity, bool addUnfilledOrderToOrderbook)
  - function makeSellOrderERC1155(address token, uint256 tokenId, uint256 powerTokenQuantity, uint256 tokenQuantity, bool addUnfilledOrderToOrderbook)
  - function makeBuyOrderERC20(address token, uint256 powerTokenQuantity, uint256 tokenQuantity, bool addUnfilledOrderToOrderbook)
  - function makeBuyOrderERC1155(address token, uint256 tokenId, uint256 powerTokenQuantity, uint256 tokenQuantity, bool addUnfilledOrderToOrderbook)
  - function cancelOrders(uint256[] calldata orderIds)

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RECC	Redundant External Contract Calls	Unresolved
●	CO	Code Optimization	Unresolved
●	RC	Redundant Calculations	Unresolved
●	ZD	Zero Division	Unresolved
●	UF	Unimplemented Function	Unresolved
●	OO	OrderId Override	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved

●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

RECC - Redundant External Contract Calls

Criticality	Minor / Informative
Location	AuctionHouseMainFacet.sol#L161,190,192,198,204
Status	Unresolved

Description

Redundant contract external calls refers to instances in a Solidity smart contract where external contract calls are made in a way that may cause unnecessary gas consumption, potential security vulnerabilities, and/or blockchain bloat. This can be caused by calling external contracts in a loop without proper checks for gas limits or reentrancy attacks, or by executing multiple redundant calls to the same contract.

The contract executes the `_executeOrder` function in a loop, which internally calls external contracts to transfer tokens to a recipient. The parameters at the code segments below are always the same on each iteration based on the case(buy/sell). The only parameters that change are the amounts to be transferred. As a result, the transaction may consume a lot of gas.

```
_executeOrder(bestOrderId, quantityToBuy);
...
_distributeTokensDirect(sellerFee);
TokenFeesStorage.layout().powerToken.transfer(
    msg.sender,
    (price * _quantity) / PRICE_FACTOR - sellerFee
);

_distributeTokens(msg.sender, sellerFee + calcFee(token, Side.BUY, (price *
_quantity) / PRICE_FACTOR));
_safeTransferOut(token, tokenId, msg.sender, _quantity, isERC20);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

A recommended approach would be to return a tuple from the `_executeOrder` function with the amounts to be transferred for each case. These amounts could be accumulated inside the loop, and when the contract exits the loop, the accumulated amounts would be transferred only once, depending on the case.

SAO - Storage Access Optimization

Criticality	Minor / Informative
Location	AuctionHouseBase.sol#L139
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The function `getBestOrder` retrieves information for an OrderBook, from a deeply nested mapping, more than once. The order book information could be searched only once and reused where needed.

```
function getBestOrder(address _token, uint256 _tokenId, Side _side) public view
returns (uint256, uint256) {
    uint256 price = _side == Side.BUY
        ? markets[_token][_tokenId][_side].priceTree.last()
        : markets[_token][_tokenId][_side].priceTree.first();
    uint256 bestOrderId;
    (, bestOrderId, ) =
markets[_token][_tokenId][_side].orderList[price].getNode(0);
    return (bestOrderId, price);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	AuctionHouseMainFacet.sol#L182,194,198,202,222,226
Status	Unresolved

Description

Redundant calculations can occur in various forms in a smart contract, such as repetitive calculations for the same result, the recalculation of unchanging values, and the repeated execution of complex computations. These redundant calculations can significantly increase the gas cost of executing the smart contract. The contract executes redundant calculations in the segments listed below.

- `_executeOrder()`: The operations `(price * _quantity) / PRICE_FACTOR` is calculated more than once.
- `_addOrder()`: The operation `(_price * _quantity) / PRICE_FACTOR` is calculated twice.

```
uint256 sellerFee = calcFee(token, Side.SELL, (price * _quantity) / PRICE_FACTOR);
TokenFeesStorage.layout().powerToken.transfer(
    msg.sender,
    (price * _quantity) / PRICE_FACTOR - sellerFee
);
...
```

Recommendation

The team is advised to take into consideration these segments and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

ZD - Zero Division

Criticality	Minor / Informative
Location	AuctionHouseMainFacet.sol#L139
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
uint256 price = (PRICE_FACTOR * _powerTokenQuantity) / _tokenQuantity;
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow executing of the corresponding statements.

OO - OrderId Override

Criticality	Minor / Informative
Location	AuctionHouseInit.sol#L15
Status	Unresolved

Description

The `AuctionHouseInit` contract is an initialization contract. The `init` function initializes the supported interfaces, grants roles and assigns an initial value to `nextOrderId`.

The `nextOrderId` is used as an identifier for orders used at the rest of the contracts, most notably at the `AuctionHouseMainFacet`. Additionally, the `init` function can be called more than once and by any user without restrictions. If a user reinitializes the auction house with a `nextOrderId` value smaller than the contract's current, then several orders will be overriden.

```
function init(address _powerTokenAddress, uint256 _nextOrderId) external {
    ...
    nextOrderId = _nextOrderId;
}
...
orders[nextOrderId] = Order({
    orderId: nextOrderId,
    side: _side,
    token: _token,
    tokenId: _tokenId,
    isERC20: _isERC20,
    sender: _sender,
    price: _price,
    quantity: _quantity
});
```

Recommendation

The team is advised to add control parameter so that the `init` function will not be able to be called more than once and it should be called only by authorized users.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	AuctionHouseGettersFacet.sol#L157,206,241
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
OrderBook storage orderBook
LinkedListLibrary.LinkedList storage orderList
OrderBook storage orderBook
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions, because it is unnecessary and can make the code less readable.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	AuctionHouseBase.sol#L87
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public nextOrderId
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	AuctionHouseInit.sol#L15 AuctionHouseGettersFacet.sol#L19,34,35,36,37,68,69,70,88,89,90,91,123,141,154,181,182,183,201,202,203,204,226,239 AuctionHouseBase.sol#L126,139,157 AuctionHouseAdminFacet.sol#L26,39,46
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _powerTokenAddress
uint256 _nextOrderId
Side _side
address _token
uint256 _price
uint256 _tokenId
uint256 _quantity
address _user
uint256 _fee
address[] memory _feeAddresses
uint256[] memory _feePercents
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	AuctionHouseBase.sol#L83,89
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
mapping(address => LinkedListLibrary.LinkedList) userOrders
uint256 internal constant PRICE_FACTOR = 1000000000
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	AuctionHouseBase.sol#L113
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _oppositeSide(Side _side) internal pure returns (Side) {
        return Side(1 - uint8(_side));
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	AuctionHouseMainFacet.sol#L139,140
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 price = (PRICE_FACTOR * _powerTokenQuantity) / _tokenQuantity
require(
            _side == Side.BUY
                ? price * _tokenQuantity <= _powerTokenQuantity * PRICE_FACTOR
                : price * _tokenQuantity >= _powerTokenQuantity * PRICE_FACTOR,
            "Rounding problem"
        )
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	AuctionHouseMainFacet.sol#L319 AuctionHouseGettersFacet.sol#L42,96,97
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bytes memory data
uint256 quantity
uint256 cost
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	AuctionHouseMainFacet.sol#L3 AuctionHouseInit.sol#L2 AuctionHouseGettersFacet.sol#L3 AuctionHouseBase.sol#L3 AuctionHouseAdminFacet.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.6;
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	AuctionHouseMainFacet.sol#L117,192,199,223,299,317
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
TokenFeesStorage.layout().powerToken.transfer(msg.sender, (price * quantity) /
PRICE_FACTOR)

TokenFeesStorage.layout().powerToken.transfer(
                    msg.sender,
                    (price * _quantity) / PRICE_FACTOR - sellerFee
                )

TokenFeesStorage.layout().powerToken.transferFrom(
                    msg.sender,
                    sender,
                    (price * _quantity) / PRICE_FACTOR - sellerFee
                )

...
```

Recommendation

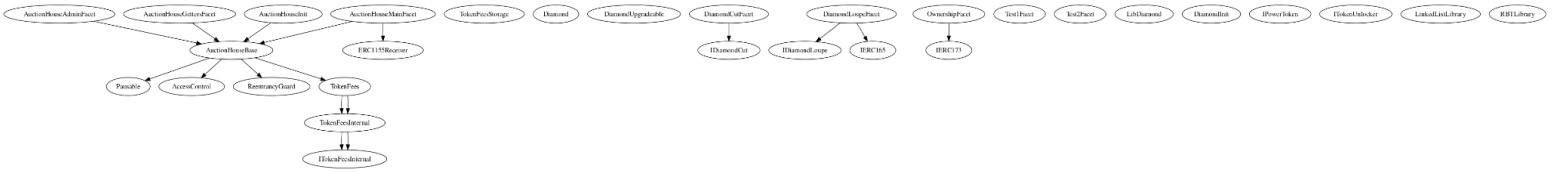
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AuctionHouse AdminFacet	Implementation	AuctionHouseBase		
	pause	External	✓	onlyRole
	unpause	External	✓	onlyRole
	setDefaultFee	External	✓	onlyRole
	setMarketFee	External	✓	onlyRole
	setFees	External	✓	onlyRole
AuctionHouse Base	Implementation	Pausable, AccessControl, ReentrancyGuard, TokenFees		
	_oppositeSide	Internal		
	getBestOrder	Public		-
	getBestOrder	Public		-
	calcFee	Public		-
AuctionHouse GettersFacet	Implementation	AuctionHouseBase		
	getQuantityUpToPrice	External		-
	getQuantityUpToPrice	Public		-
	getCostForQuantity	External		-
	getCostForQuantity	Public		-
	getUserOpenOrders	External		-
	getPrices	Public		-
	getPrices	Public		-

	getOrdersAtPrice	Public		-
	getOrdersAtPrice	Public		-
	getNumPrices	Public		-
	getNumPrices	Public		-
AuctionHouseInit	Implementation	AuctionHouseBase		
	upgrade	External	✓	-
	init	External	✓	-
AuctionHouseMainFacet	Implementation	AuctionHouseBase, ERC1155Receiver		
	makeSellOrderERC20	External	✓	whenNotPaused
	makeSellOrderERC1155	External	✓	whenNotPaused
	makeBuyOrderERC20	External	✓	whenNotPaused
	makeBuyOrderERC1155	External	✓	whenNotPaused
	cancelOrders	External	✓	nonReentrant
	_makeOrder	Internal	✓	
	_executeOrder	Internal	✓	
	_addOrder	Internal	✓	
	_removeOrder	Internal	✓	
	_safeTransferIn	Internal	✓	
	_safeTransferOut	Internal	✓	
	_safeTransferFrom	Internal	✓	
	_balanceOf	Internal		

Inheritance Graph



Flow Graph

Summary

DeFi Kingdoms contracts implement a financial mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io