Cyberscope

Audit Report

Trait Exchange

October 2023

Repository	https://github.com/prshnandaniya/Trait-Smart-Contract/blob/main/NFTSwap.sol
Commit	9d10f69ffc31ddf903f6d902880c8cc9403b05b3
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/prshnandaniya/Trait-Smart-Contract/blob/main/NFTSwap.sol
Commit	9d10f69ffc31ddf903f6d902880c8cc9403b05b3
Testing Deploy	https://testnet.bscscan.com/address/0xd26685c914ef5fae7b24b0e12fcc079982859c8a

Audit Updates

Initial Audit	03 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/trait/v1/audit.pdf
Corrected Phase 2	18 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/trait/v2/audit.pdf
Corrected Phase 3	23 Oct 2023

Source Files

Filename	SHA256
contracts/Trait.sol	0fdb1f792c8b64d2a199e61fb4946ff81c2469262fecaf1d483705fb7a2f26e8
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5

@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	77f0f7340c2da6bb9edbc90ab6e7d3eb8e2ae18194791b827a3e8c0b11a09b43
@openzeppelin/contracts/token/ERC721/IERC721.sol	c8d867eda0fd764890040a3644f5ccf5db92f852779879f321ab3ad8b799bf97
@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol	995380c950997080660556ba6bc4e3bb37c1323b0539f31d6af40349fb2d7c4c
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol	578834a1bcdac6a22de5e07ae63bbbd4d41615f35950afc6e6c068d92619b334
@openzeppelin/contracts/token/ERC1155/IERC1155.sol	11a1673048905dbd7703b3e23c22d612484db9c0b5a629c303ed609c04850f3b
@openzeppelin/contracts/token/ERC1155/utils/ERC1155Receiver.sol	cf407886a0ce7e2af7efe7867e2d2864903426f63eeaa68eecf33d57f7d910c2
@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol	a7ad38fa0a06fe6e24f81fee4f1fc3870767db96d1ba37df7be1199f7a3ace7f
@openzeppelin/contracts/access/Ownable.sol	a8e4e1ae19d9bd3e8b0a6d46577eec098c01fbaffd3ec1252fd20d799e73393b

Overview

The `Trait` contract facilitates a decentralized marketplace for trading assets on the blockchain. Users can create offers, specifying the assets they wish to exchange, including native token (i.g. ETH), ERC20 tokens, ERC721 and ERC1155 tokens (NFTs). The contract handles the transfer of these assets between parties, ensuring that the terms of the offer are met. Key features include the ability to offer multiple types of assets in a single transaction, validation checks to ensure the correct amount of ETH is sent, and mechanisms for offer management, such as rejection and withdrawal. The contract also incorporates fee handling, with certain users who are NFT holders of a specific collection being excluded from these fees. The design aims to provide a secure and efficient platform for peer-to-peer asset trading.

Create Offer Functionality

The `createOffer` function allows users to create offers on the Trait contract. This function is designed to facilitate the exchange of assets between users. Here's a breakdown of its functionality:

Parameters Specification: Users specify the parameters of the Struct `_structOffer` they want to create. These parameters include:

- The recipient of the offer (_receiver).
- The amount of Ethereum they are offering (_offeredETH).
- The amount of Ethereum they are requesting in return (_requestedETH).
- The ERC20 tokens they are offering (_offeredERC20).
- The ERC20 tokens they are requesting in return (_requestedERC20).
- The NFTs (Non-Fungible Tokens) they are offering (_offeredERC721).
- The NFTs they are requesting in return (_requestedERC721).
- The ERC1155 NFTs they are offering (_offeredERC1155).
- The ERC1155 NFTs they are requesting in return (_requestedERC1155).
- The duration for which the offer is valid (_offerValidDuration).

The `createOffer` function authorizes users to create offers within the smart contract. When invoking this function, users specify the parameters of their offer, including the desired NFT (ERC721 or ERC1155), ERC20 tokens, and ETH amount. Concurrently, they also specify the assets they are willing to offer in exchange, which could be an NFT (ERC721 or ERC1155), ERC20 tokens, or a certain ETH value. These offered assets are then transferred from the user's account (msgSender) to the contract. If a user isn't a holder of an NFT that is listed in the excluded from fees contracts, they are obligated to pay an additional fee ammount. Upon successful validation and fee payment, the offer is officially registered in the contract, with both the proposed and requested assets recorded.

Accept Offer Functionality

The `acceptOffer` function of the Trait contract, enables users to accept offers made by other participants. Upon invoking the `acceptOffer` function, users are required to provide a valid `_offerId` which is then checked against the existing offers to ensure its validity. The function ensures that the individual accepting the offer (`msgSender`) is indeed the intended receiver of the offer. Additionally, it verifies that the ETH sent by the receiver matches or exceeds the requested ETH amount in the offer. A time check confirms that the offer hasn't expired based on its creation timestamp and valid duration.

Exchange of assets:

The function facilitates the transfer of offered ERC721 and ERC1155 tokens from the contract to the receiver. Conversely, the requested ERC721 and ERC1155 tokens are transferred from the receiver to the offer creator. If there's any ETH offered, it's directly transferred to the receiver's account. Similarly, any requested ETH is sent to the offer creator's account. For ERC20 tokens, if any are requested in the offer, they are transferred from the receiver to the offer creator. Conversely, if any ERC20 tokens are offered, they are transferred to the receiver.

Once all asset transfers are successfully executed, the status of the offer is updated to `accepted` , and an event is emitted to log this change in offer status. Throughout the process, it's imperative for users to ensure they've set adequate allowances, to avoid any transaction failures.

Reject Offer Functionality

The `rejectOffer` function give the ability to users to decline offers.

When a user decides to invoke the `rejectOffer` function, they must provide a valid `_offerId`. This ID is then cross-referenced with the existing offers in the contract to confirm its authenticity. The function strictly ensures that the individual rejecting the offer (msgSender) is the intended receiver of that particular offer in order to prevent unauthorized rejections. Upon successful validation, the status of the offer is updated to `rejected`.

Withdraw Offer Functionality

The `withdrawOffer` function, grant users the ability to retract offers they've previously made. When a user wishes to withdraw an offer, they initiate the `withdrawOffer` function by providing the relevant `_offerId`. This ID is then matched with the existing offers in the contract to ascertain its validity. A pivotal aspect of this function is that only the original creator of the offer (msgSender) can execute the withdrawal. This safeguard ensures that only authorized users can retract offers, preventing potential misuse.

Upon successful validation, all the offered ERC721 and ERC1155 tokens are returned from the contract back to the original offer creator. If there's any ETH that was part of the offer, it's transferred back to the offer creator's account. For ERC20 tokens, if any were included in the offer, they are also returned to the offer creator. The transfer is executed only if the ERC20 token contract address is valid and the offered ERC20 token value is greater than zero. Once all assets are reverted back to the offer creator, the status of the offer is updated to `withdrawan`.

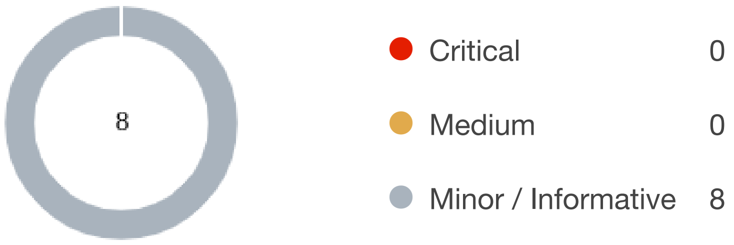
Owner Functionalities

The owner of the contract has the ability to set specific NFT contract addresses that are exempted from incurring exchange fees. This is achieved through the `excludeFromExchangeFees` function, where the owner can input a contract address to be exempted, ensuring that holders of the NFTs from that particular contract are not burdened with additional fees.

Additionally, the owner can invoke the `setFees` function to modify the fee amount, ensuring flexibility in fee management based on evolving requirements or market conditions.

When the owner decides to claim the accumulated fees, the `claimFees` function facilitates this process. It calculates the pending fees, updates the claimed fees record, and transfers the pending amount directly to the owner's account. This action is then transparently recorded through the `FeesClaimedByAdmin` event, ensuring traceability and transparency in the contract's financial operations.

Findings Breakdown



	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	8	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	MMI	Modifier Memory Inefficiency	Unresolved
●	RMC	Redundant Modifier Checks	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MU	Modifiers Usage	Unresolved
●	ITT	Inconsistent Token Transfer	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

MMI - Modifier Memory Inefficiency

Criticality	Minor / Informative
Location	contracts/Trait.sol#L94,116,136
Status	Unresolved

Description

The contract contains the `isOfferValidForWithdrawal`, `isValidOffer` and `isReceiver` modifiers that fetch the offerAccount struct from the `_mappingOffer` mapping. These modifiers utilize the `memory` keyword when declaring the `offerAccount` variable. While the `memory` keyword is suitable for temporary data storage, it's not optimal when only a few attributes of a struct are accessed, as it loads the entire struct into `memory`, thereby increasing gas costs.

```
  modifier isOfferValidForWithdrawal(uint256 _offerId) {
        StructOffer memory offerAccount = _mappingOffer[_offerId];

        require(
            offerAccount.sender != address(0),
            "Address zero cannot make offer."
        );

        ...

        _;
    }

    modifier isValidOffer(uint256 _offerId) {
        StructOffer memory offerAccount = _mappingOffer[_offerId];

        require(
            offerAccount.sender != address(0),
            "Address zero cannot make offer."
        );

        ...
        _;
    }

    modifier isReceiver(uint256 _offerId) {
        StructOffer memory offerAccount = _mappingOffer[_offerId];
        require(
            msg.sender == offerAccount.receiver,
            "You are not the receiver of this offer."
        );
        _;
    }
```

Recommendation

It is recommended to use the `storage` keyword inside the modifiers instead of `memory`. By referencing the struct directly in `storage`, the contract can access only the necessary attributes, leading to potential gas savings. This change will make the contract more gas-efficient, especially during frequent interactions.

RMC - Redundant Modifier Checks

Criticality	Minor / Informative
Location	contracts/Trait.sol#L116,150
Status	Unresolved

Description

The contract contains the `createOffer` function that allows users to create offers. Within this function, the require statement ensures that the receiver's address is not the zero address. Additionally, the contract employs the `isValidOffer` modifier, which checks if both the sender and receiver addresses are not the zero address. However, since the `createOffer` function already checks for the receiver's address being the zero address and the sender's address is inherently the `msg.sender`, the first two require statements inside the `isValidOffer` modifier are redundant.

```
    modifier isValidOffer(uint256 _offerId) {
        StructOffer memory offerAccount = _mappingOffer[_offerId];

        require(
            offerAccount.sender != address(0),
            "Address zero cannot make offer."
        );

        require(
            offerAccount.receiver != address(0),
            "Cannot make offer to address zero."
        );

        require(
            offerAccount.status == OfferStatus.pending,
            "Offer already accepted or withdrawan."
        );
        _;
    }

    function createOffer(StructOffer memory _structOffer)
        external
        payable
        returns (uint256 offerId)
    {
        require(
            _structOffer.receiver != address(0),
            "createOffer(): _receiver cannot be address zero."
        );
    ...
    address msgSender = msg.sender;
    ....
    offerAccount.sender = msgSender;
    ....
}
```

Recommendation

It is recommended to retain only one of the two non-zero address checks within the `isValidOffer` modifier, either for the sender or the receiver, alongside with the check that ensures the offer's status in a pending state.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/Trait.sol#L168
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

Specifically, within the `createOffer` function, the same error message is used for the requires on the `offeredERC721` , `offeredERC1155` , and `requestedERC1155` checks. The error comment intended for the `offeredERC721` is being used for both the `offeredERC1155` and `requestedERC1155` tokens. This can lead to confusion and misinterpretation of the error, especially when debugging or when users encounter this error.

```
require(
    _structOffer.offeredERC721.length < type(uint8).max,
   "createOffer(): Offered erc721 cannot be more than 255"
);
...
require(
    _structOffer.offeredERC1155.length < type(uint8).max,
    "createOffer(): Offered erc721 cannot be more than 255"
);
require(
    _structOffer.requestedERC1155.length < type(uint8).max,
    "createOffer(): Offered erc721 cannot be more than 255"
);
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract. It is recomended to update the error messages to reflect the actual token types being checked and the `Offered` or `requested` state. This will improve clarity and reduce potential misunderstandings.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/Trait.sol#L150
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

Specifically the contract is using a `require` statement within the `createOffer` function to ensure that the `receiver` address is not the zero address, instead of using the `isReceiver` modifier.

```
require(
     _structOffer.receiver != address(0),
    "createOffer(): _receiver cannot be address zero."
);
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain. It is recomended to use the `isReceiver` modifier in the `createOffer` function.

ITT - Inconsistent Token Transfer

Criticality	Minor / Informative
Location	contracts/Trait.sol#L317
Status	Unresolved

Description

The contract facilitates the transfer of both ERC721 and ERC1155 tokens. However when an offer is accepted, the contract transfers the requested ERC1155 token from the `offerAccount.receiver` to the `offerAccount.sender` using the `"0x"` value as the data. This behavior is inconsistent with other ERC1155 token transfers within the contract, which utilize the data field of the `offerAccount.offeredERC1155` structure. This inconsistency can lead to unexpected behaviors, especially if the receiving end of the transfer expects certain data to be associated with the token transfer.

```
   function acceptOffer(uint256 _offerId)
        external
        payable
        noReentrancy
        isValidOffer(_offerId)
        isReceiver(_offerId)
    {
        ...
        for (uint8 i; i < offerAccount.offeredERC1155.length;
i++) {

IERC1155(offerAccount.offeredERC1155[i].erc1155Contract)
                .safeTransferFrom(
                    address(this),
                    offerAccount.receiver,
                    offerAccount.offeredERC1155[i].erc1155Id,
                    offerAccount.offeredERC1155[i].amount,
                    offerAccount.offeredERC1155[i].data
                );
        }
        ...
        for (uint8 i; i < offerAccount.requestedERC1155.length;
i++) {

IERC1155(offerAccount.requestedERC1155[i].erc1155Contract)
                .safeTransferFrom(
                    offerAccount.receiver,
                    offerAccount.sender,
                    offerAccount.requestedERC1155[i].erc1155Id,
                    offerAccount.requestedERC1155[i].amount,
                    "0x"
                );
        }
    ...
}
```

Recommendation

It is recommended to streamline the code to ensure consistent handling of ERC1155 token transfers. Specifically, the data field from the offerAccount.requestedERC1155 or offeredERC1155 structure could be used in all ERC1155 transfers, rather than hardcoding the `"0x"` value. This will ensure that all ERC1155 transfers within the contract behave in a consistent manner, reducing the potential for errors or unexpected behaviors.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Trait.sol#L145,259,355,367,419,427,469,488,510
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.

```
StructOffer memory _structOffer
uint256 _offerId
address _userAddress
address _contractAddress
uint256 _feesInWei
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/Trait.sol#L214,224,228,241,281,289,301,310,381,389,453,473,495
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.

```
uint8 i
```

Recommendation

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

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/Trait.sol#L205,333,345,408
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.

```
IERC20(_structOffer.offeredERC20.erc20Contract).transferFrom(
                msgSender,
                address(this),
                _structOffer.offeredERC20.erc20Value
            )

...
                offerAccount.requestedERC20.erc20Value
            )

IERC20(offerAccount.offeredERC20.erc20Contract).transfer(
                offerAccount.receiver,
                offerAccount.offeredERC20.erc20Value
            )

...
```

Recommendation

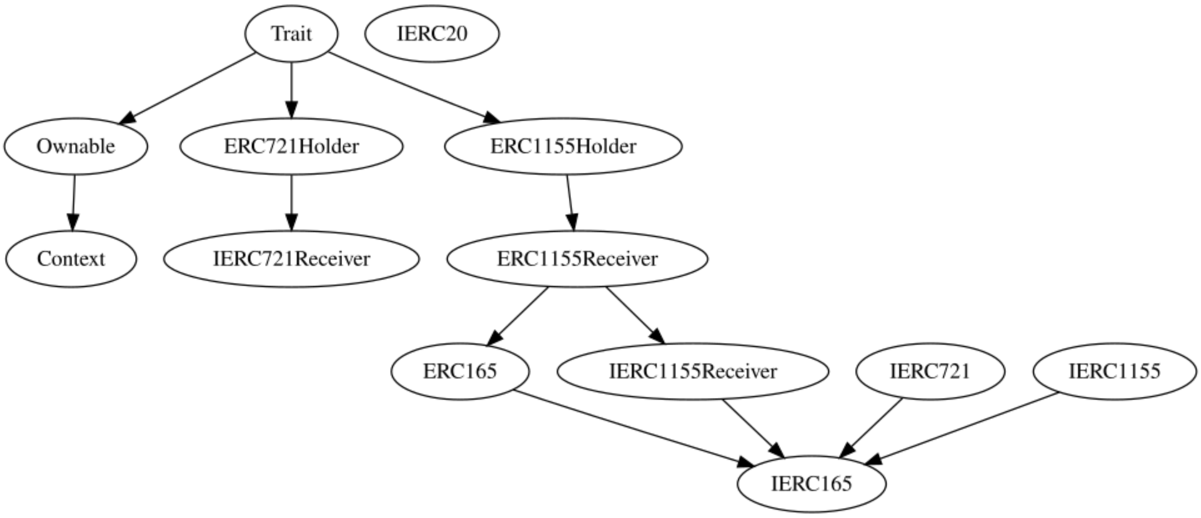
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

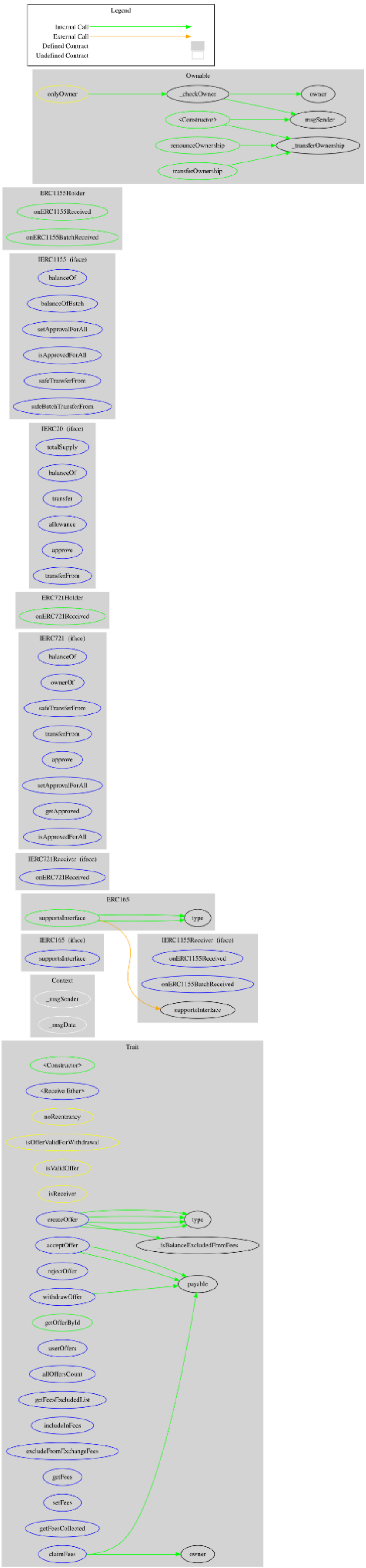
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Trait	Implementation	Ownable, ERC721Holder, ERC1155Holder		
		Public	✓	-
		External	Payable	-
	createOffer	External	Payable	-
	acceptOffer	External	Payable	noReentrancy isValidOffer isReceiver
	rejectOffer	External	✓	noReentrancy isValidOffer isReceiver
	withdrawOffer	External	✓	noReentrancy isOfferValidForWithdrawal
	getOfferById	Public		-
	userOffers	External		-
	allOffersCount	External		-
	_isBalanceExcludedFromFees	Private		
	getFeesExcludedList	External		-
	includeInFees	External	✓	onlyOwner
	excludeFromExchangeFees	External	✓	onlyOwner
	getFees	External		-

	setFees	External	✓	onlyOwner
	getFeesCollected	External		-
	claimFees	External	✓	noReentrancy onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
ERC1155Holder
onERC1155Received
onERC1155BatchReceived
IERC1155 (iface)
balanceOf
balanceOfBatch
setApprovalForAll
isApprovedForAll
safeTransferFrom
safeBatchTransferFrom
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ERC721Holder
onERC721Received
IERC721 (iface)
balanceOf
ownerOf
safeTransferFrom
transferFrom
approve
setApprovalForAll
getApproved
isApprovedForAll
IERC721Receiver (iface)
onERC721Received
ERC165
supportsInterface
type
IERC165 (iface)
supportsInterface
IERC1155Receiver (iface)
onERC1155Received
onERC1155BatchReceived
supportsInterface
Context
_msgSender
_msgData
Trait
<Constructor>
<Receive Ether>
noReentrancy
isOfferValidForWithdrawal
isValidOffer
isReceiver
createOffer
type
acceptOffer
isBalanceExcludedFromFees
rejectOffer
payable
withdrawOffer
getOfferById
userOffers
allOffersCount
getFeesExcludedList
includeInFees
excludeFromExchangeFees
getFees
setFees
getFeesCollected
claimFees
owner

Summary

Trait Exchange contract facilitates a decentralized trading platform for digital assets on the blockchain. 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