Cyberscope

Audit Report

Escrow

March 2023

SHA256 814214762f47f261e5d115db1eca1b3704160db5ad2d7dd53282a1ba82417e35

Audited by © cyberscope

Table of Contents

Review

Contract Name	Escrow
Testing Deploy	https://testnet.bscscan.com/address/0x9c9c2763b6e4e393c56cb6b5ef24f89c570856b6

Audit Updates

Initial Audit	13 Feb 2023
Corrected Phase 2	15 Mar 2023

Source Files

Filename	SHA256
@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol	ab7e1f18f36b72e9ab184eb33933d4aa1b5de3f1ca359b56fc6cfabccad952d1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
contracts/Escrow.sol	814214762f47f261e5d115db1eca1b3704160db5ad2d7dd53282a1ba82417e35

Introduction

The Escrow contract implements an escrow mechanism, which presents three distinct categories for escrow items: product, service, and crypto trade.

Escrow Items

Service and Product

The product and service items are similar, with the only difference being that the service item has a specified delivery duration. Only the buyer can request an escrow service for either item, and only the owner and a manager can resolve any conflicts that arise to ensure the completion of the service.

For the service and product categories to operate properly, they must follow the same procedures.

1. The buyer has to create a trade.
2. The seller has to deliver the product.
3. The buyer has to check the product and then release the fund.

Product/Service States

The escrow categories consist of three states.

- AWAITING_DELIVERY
- AWAITING_FUND_RELEASE
- COMPLETE

Crypto Trade

The crypto trade category places a crypto asset for trade. A crypto trade has to follow two steps.

1. The token owner has to create the crypto trade.
2. Any buyer can buy the corresponding crypto trade in order to be completed.

Roles

The contract consists of the owner and the manager roles.

The `Owner` role has the authority to

- Set taxes.
- Grant or revoke the manager role.
- Manages ban addressed.
- Resolve an item appeal.

The `Manager` role has the authority to

- Resolve an item appeal

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RAV	Reentrancy Attack Vulnerability	Unresolved
●	DDS	Duplicate Data Structure	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	CR	Code Repetition	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	BA	Ban Addresses	Unresolved
●	CO	Code Optimization	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L19	Stable Compiler Version	Unresolved

●	L20	Succeeded Transfer Check	Unresolved

RAV - Reentrancy Attack Vulnerability

Criticality	Critical
Location	contracts/Escrow.sol
Status	Unresolved

Description

The contract is vulnerable to a reentrancy attack, which can occur if a buyer initiates a trade using a contract address as the seller or by initiating a crypto trade from a contract. For instance,

- A crypto buyer creates and buys a crypto trade with a contract address as the seller's address.
- Then the seller will execute the function buyCrypto which internally calls the (seller).transfer(payAmount) method.
- A reentrance attack will occur if the seller implements a receive callback, they will have the ability to execute buyCrypto method again within the same execution thread.

```
function buyCrypto (uint256 cryptoTradeId) external payable{
    CryptoItem storage cryptoProduct = cryptoTrades[cryptoTradeId];
    require(msg.value >= cryptoProduct.price, "Not enough paid");
    require(!cryptoProduct.completed, "Already completed");

    // tax policy
    (cryptoProduct.seller).transfer(cryptoProduct.price);
```

Recommendation

The contract could incorporate a mutex pattern modifier or disallow the use of contract addresses as the seller address.

DDS - Duplicate Data Structure

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L10,25
Status	Unresolved

Description

The contract utilized two data structures `ProductItem` and `ServiceItem` to illustrate almost the same information.

```
struct ProductItem
{
    string chatRoomNumber;
    string productName;
    string productLink;
    address payable buyer;
    address payable seller;
    uint256 price;
    STATE currentState;
    uint256 createTime;
    uint256 deliverTime;
    bool fundsWithdrawn;
    bool appeal;
}

struct ServiceItem
{
    string chatRoomNumber;
    string serviceName;
    string serviceLink;
    address payable buyer;
    address payable seller;
    uint256 price;
    STATE currentState;
    uint256 createTime;
    uint256 duration;
    uint256 deliverTime;
    bool fundsWithdrawn;
    bool appeal;
}
```

Recommendation

The team is advised to merge the two data structures. That way it will enhance the efficiency, readability, and performance of the source code, while also decreasing the cost of executing it.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L64,65,66
Status	Unresolved

Description

The contract is using the variable `currentProductTradeId`, `currentServiceTradeId` and `currentCryptoTradeId` to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint256 public currentProductTradeId;
uint256 public currentServiceTradeId;
uint256 public currentCryptoTradeId;
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L140,180,207,255,280
Status	Unresolved

Description

The contract contains repetitive code segments. The contract duplicates the same code segment for the two data structures `ServiceItem` and `ProductItem`. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
function createNewTrade (string memory _chatRoomNumber, string memory
_productName, uint256 _price, address _seller, uint256 _duration,
uint256 category) external payable

function deliverProduct(uint256 tradeId, string memory _productLink,
uint256 category) external

function releaseFunds(uint256 tradeId, uint256 category) external

function appeal(uint256 tradeId, uint256 category) external

function resolveAppeal(uint256 tradeId, bool buyerWin, uint256
category) external onlyManager
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help to reduce the complexity and size of the contract. For instance, the contract could merge the two data structures in order to avoid repeating the same code in multiple places.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L234
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

The tax `currentPrice * totalTax` may not be splitted as expected.

```
function payTax(uint256 price, bool success) internal returns(uint256
payAmount) {
    uint256 currentPrice = getLatestPrice();

    (teamWallet1).transfer(currentPrice * totalTax * 125 / 1000); //
12.5%
    (teamWallet2).transfer(currentPrice * totalTax * 125 / 1000); //
12.5%
    (teamWallet3).transfer(currentPrice * totalTax * 125 / 1000); //
12.5%
    (teamWallet4).transfer(currentPrice * totalTax * 20 / 100);    //
20%
    (teamWallet5).transfer(currentPrice * totalTax * 375 / 1000); //
37.5%
    (teamWallet6).transfer(currentPrice * totalTax * 5 / 100);     // 5%

    uint256 tax7;
    if (success)
        tax7 = price / 100;
    else
        tax7 = price / 200;
    payAmount = price - currentPrice * totalTax - tax7;
}
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

BA - Ban Addresses

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L129
Status	Unresolved

Description

The contract owner has the authority to ban addresses from creating new Escrow requests. The owner can ban addresses by calling the `manageBannedAddress` function.

```
function manageBannedAddress(address illegalAddress, bool isAdd)
external onlyOwner {
     if (isAdd)
         bannedAddresses[illegalAddress] = true;
     else
         bannedAddresses[illegalAddress] = false;
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L393
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.

Since the token address is saved on the currency variable the decimals variable is redundant.

```
cryptoProduct.currency = _currencyAddress;
cryptoProduct.decimals = token.decimals();
```

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. It is recommended to avoid duplicating data that is already accessible from existing sources.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L162,163,164
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
priceFeedAddres
priceFee
owne
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L133,134,135,136,137,138,167,203,243,442
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 payable public constant teamWallet1 =
payable(0x4b4a0CBB2A7c971D51Ae7dE040a7a290498Df74E)
address payable public constant teamWallet2 =
payable(0xCb10616fDfd7a5f3e3e144Aad8e7D7821DFAb6A2)
address payable public constant teamWallet3 =
payable(0x936A0cA35971Fe8A48000829f952e41293ea0DC8)
address payable public constant teamWallet4 =
payable(0x595F21963feDbc4f5BA4A11b76359dEe916040c0)
address payable public constant teamWallet5 =
payable(0xd136EB70B571cEf8Db36FAd5be07cB4F76905B64)
address payable public constant teamWallet6 =
payable(0xd136EB70B571cEf8Db36FAd5be07cB4F76905B64)
uint256 _totalTax
uint256 _price
string memory _chatRoomNumber
uint256 _duration
string memory _productName
address _seller
string memory _productLink
address _currencyAddress

...
```

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.

L07 - Missing Events Arithmetic

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

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
totalTax = _totalTax
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/Escrow.sol#L209,223,451
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.

```
ProductItem memory product
ServiceItem memory service
CryptoItem memory cryptoProduct
```

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	contracts/Escrow.sol#L2,9
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 ^0.8.12;
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	contracts/Escrow.sol#L449,476
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.

```
token.transferFrom(msg.sender, address(this), _amount)
token.transfer(msg.sender, cryptoProduct.amount)
```

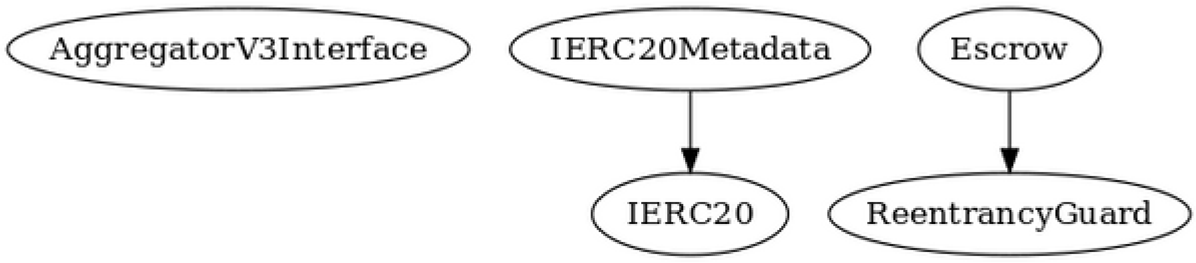
Recommendation

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.

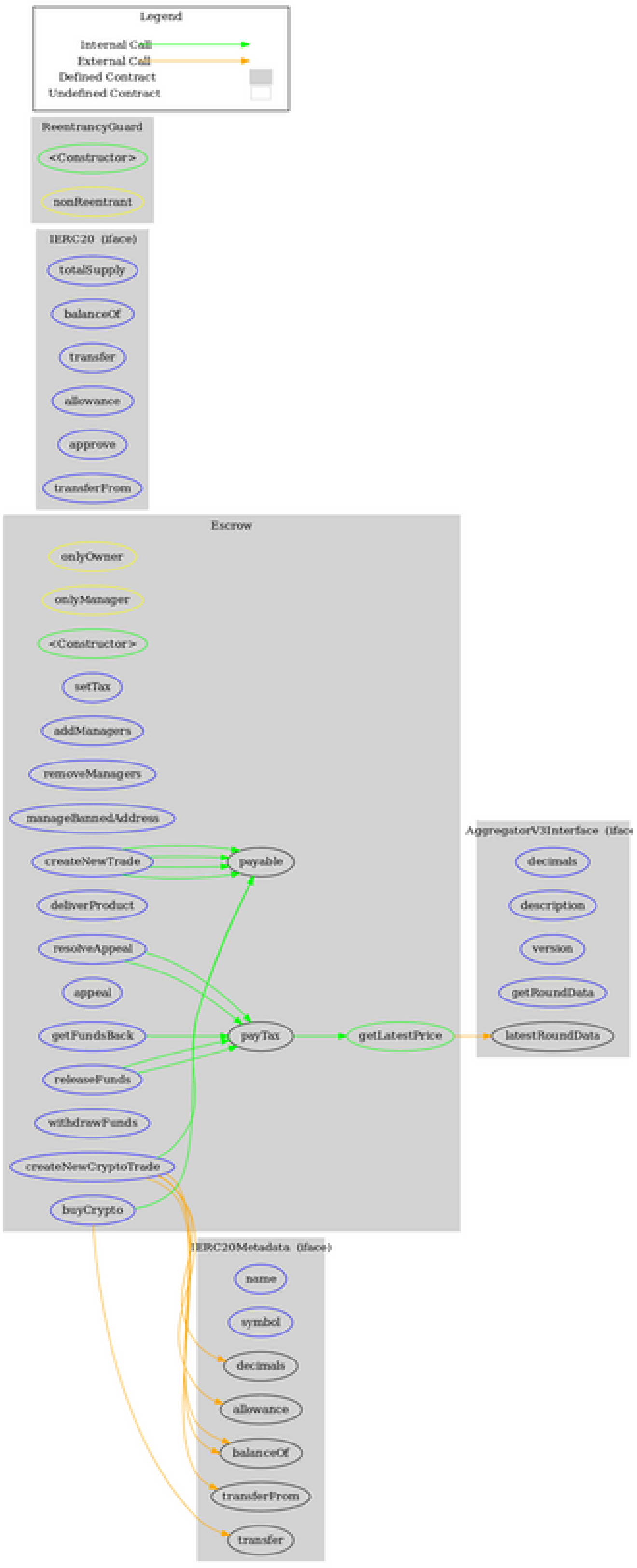
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ReentrancyGuard	Implementation			
		Public	✓	-
Escrow	Implementation	ReentrancyGuard		
		Public	✓	-
	setTax	External	✓	onlyOwner
	getLatestPrice	Public		-
	addManagers	External	✓	onlyOwner
	removeManagers	External	✓	onlyOwner
	manageBannedAddress	External	✓	onlyOwner
	createNewTrade	External	Payable	-
	deliverProduct	External	✓	-
	releaseFunds	External	✓	nonReentrant
	payTax	Internal	✓	
	appeal	External	✓	-
	resolveAppeal	External	✓	onlyManager
	getFundsBack	External	✓	-
	withdrawFunds	External	✓	-
	createNewCryptoTrade	External	✓	-
	buyCrypto	External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ReentrancyGuard
<Constructor>
nonReentrant
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Escrow
onlyOwner
onlyManager
<Constructor>
setTax
addManagers
removeManagers
manageBannedAddress
createNewTrade
payable
deliverProduct
resolveAppeal
appeal
getFundsBack
payTax
getLatestPrice
releaseFunds
withdrawFunds
createNewCryptoTrade
buyCrypto
AggregatorV3Interface (iface
decimals
description
version
getRoundData
latestRoundData
IERC20Metadata (iface)
name
symbol
decimals
allowance
balanceOf
transferFrom
transfer

Summary

Escrow contract implements a utility and financial mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

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