CodeHawks Escrow Contract - Findings Report

Table of contents

Contest Summary

Sponsor: CodeHawks Escrow Contract - Competition Details

Dates: Jul 24th, 2023 → Aug 5th, 2023

See more contest details here

Results Summary

Number of findings:

- High: 1
- Medium: 3
- Low: 0

[H-1] TITLE (Root + Impact)
Buyers Funds stuck in `Escrow` contract if arbiter is set to 0 address
Description
If the buyer sets the arbiter address to 0x0, and is not satisfied with the seller's work, the buyer's funds will be stuck in the contract forever — unless they unhappily confirm the receipt to release the funds to the seller
Impact
1)Buyer Losss funds 2)Buyer can't withdraw funds 3) Buyes can't resolve their issue with the seller
Proof of Concepts

```
solidity

constructor(
    uint256 price,
    IERC20 tokenContract,
    address buyer,
    address seller,
    address arbiter,
    uint256 arbiterFee
)

 {
    if (address(tokenContract) == address(0)) revert Escrow__TokenZeroAddress();
    if (buyer == address(0)) revert Escrow__BuyerZeroAddress();
    if (seller == address(0)) revert Escrow__SellerZeroAddress();
    if (arbiterFee >= price) revert Escrow__FeeExceedsPrice(price, arbiterFee);
    if (tokenContract.balanceOf(address(this)) < price) revert Escrow__MustDeployWithTok
    i_price = price;
    i_tokenContract = tokenContract;
    i_buyer = buyer;
    i_seller = seller;
    i_arbiter = arbiter;
    i_arbiterFee = arbiterFee;
}
```

There is no check to ensure arbiter != address(0). If 0x0 is passed, the arbiter functionality becomes permanently unusable
Recommended mitigation
1) Add Emergency Withdrawal Function for Buyer

```
solidity

function Emergencywithdraw() external onlyBuyer {

    uint256 tokenBalance = i_tokenContract.balanceOf(address(this));
    if (tokenBalance > 0) {
        i_tokenContract.safeTransfer(i_buyer, tokenBalance);
    }

}
```

[M-1] TITLE (Root + Impact)
Fee-on-Transfer Tokens Prevent Escrow Deployment
Description

If the buyer uses fee-on-transfer tokens, they couldn't deploy the Escrow contract because if they set the price, transfer the price to the Escrow contract, it will cut the fees, so the contract balance now will be decreased. So in this case:

```
solidity

if (tokenContract.balanceOf(address(this)) < price) revert Escrow__MustDeployWithTokenBa
```

The contract balance is lower than the price, so they will get the error and they can't deploy the contract also.
Impact
1) Buyer can't able to deploy the `Escrow` contract if they used fee on transfer token
Recommended mitigation
1)Use token balance insted of fixed price

```
solidity

function newEscrow(uint256 price, IERC20 tokenContract,address seller, address arbiter, 
    address computedAddress = computeEscrowAddress(
        type(Escrow).creationCode,
        address(this),
        uint256(salt),
        price,
        tokenContract,
        msg.sender,
        seller,
        arbiter,
        arbiterFee
    );
    tokenContract.safeTransferFrom(msg.sender, computedAddress, price);
    Escrow escrow = new Escrow{salt: salt}(
   --   price,
   ++   tokenContract.balanceOf(msg.sender),
        tokenContract,
        msg.sender,
        seller,
        arbiter,
        arbiterFee
    );
    if (address(escrow) != computedAddress) {
        revert EscrowFactory__AddressesDiffer();
    }
    emit EscrowCreated(address(escrow), msg.sender, seller, arbiter);
    return escrow;
}
```

[M-2] TITLE (Root + Impact)

Rebasing Tokens Prevent `Escrow` Deployment

Description

If the buyer uses rebasing tokens like AMPL, RSR, these tokens dynamically change the balance of tokens. This causes a reduction in the token balance, so the buyer's set price will be higher than the token balance, which throws the error

```
solidity

if (tokenContract.balanceOf(address(this)) < price) revert Escrow__MustDeployWithTokenBa
```

Impact

1) Buyer can't deploy `Escrow` Contract

Recommended mitigation

1)Use token balance insted of fixed price

```
solidity

function newEscrow(uint256 price, IERC20 tokenContract,address seller, address arbiter,
    address computedAddress = computeEscrowAddress(
        type(Escrow).creationCode,
        address(this),
        uint256(salt),
        price,
        tokenContract,
        msg.sender,
        seller,
        arbiter,
        arbiterFee
    );
    tokenContract.safeTransferFrom(msg.sender, computedAddress, price);
    Escrow escrow = new Escrow{salt: salt}(
   --   price,
   ++   tokenContract.balanceOf(msg.sender),
        tokenContract,
        msg.sender,
        seller,
        arbiter,
        arbiterFee
    );
    if (address(escrow) != computedAddress) {
        revert EscrowFactory__AddressesDiffer();
    }
    emit EscrowCreated(address(escrow), msg.sender, seller, arbiter);
    return escrow;
}
```

[M-3] TITLE (Root + Impact)
If any participant is blocklisted in the token, all transfers will fail and everyone loses funds
Description
If anyone (buyer, seller, or arbiter) is blocklisted in the token contract, then no one can receive their funds, and all participants lose access to the escrowed funds.

```
solidity

function resolveDispute(uint256 buyerAward) external onlyArbiter nonReentrant inState(St
    uint256 tokenBalance = i_tokenContract.balanceOf(address(this));
    uint256 totalFee = buyerAward + i_arbiterFee; // Reverts on overflow
    if (totalFee > tokenBalance) {
        revert Escrow__TotalFeeExceedsBalance(tokenBalance, totalFee);
    }

    s_state = State.Resolved;
    emit Resolved(i_buyer, i_seller);

    if (buyerAward > 0) {
        i_tokenContract.safeTransfer(i_buyer, buyerAward);
    }
    if (i_arbiterFee > 0) {
        i_tokenContract.safeTransfer(i_arbiter, i_arbiterFee);
    }
    tokenBalance = i_tokenContract.balanceOf(address(this));
    if (tokenBalance > 0) {
        i_tokenContract.safeTransfer(i_seller, tokenBalance);
    }
}
```

Impact
1) Buyer , Arbiter , Seller everyone loss their funds
Recommended mitigation
Instead of transferring tokens directly in a single function:
1) Mark the dispute as resolved.
2) Assign balances to each participant internally.

3) Let users withdraw their tokens manually using a withdraw() function.