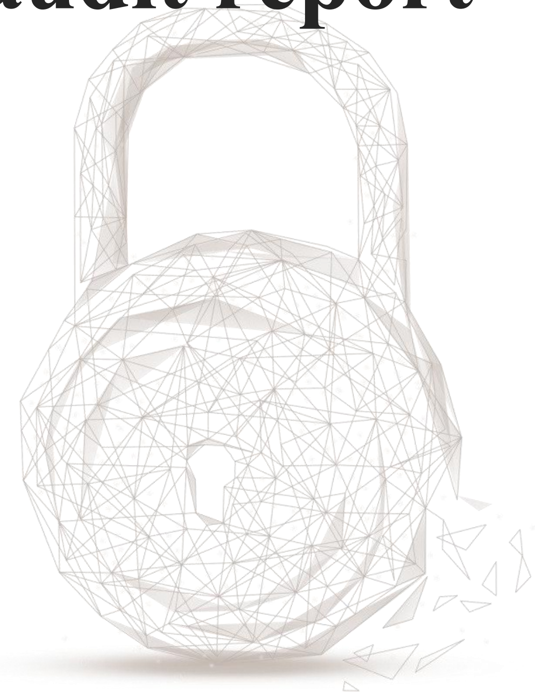
BEOSIN

Blockchain Security

# Smart contract security audit report



**Audit Number:202104091815**

**Report Query Name: BLES**

**Github link:**

https://github.com/BlindBoxesNFT/blindboxes-contracts

**Commit hash:**

3978e1464d02b00f3f83afb2dadf5cca6c67c93d

b4a9c0db54923745714c7e27582f2f8ee4dbe6f3

**Start Date:2021.04.05**

**Completion Date:2021.04.09**

**Overall Result:Pass**

**Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.**

## Audit Categories and Results:

| No. | Categories | Subitems | Results |
|---|---|---|---|
| 1 | Coding Conventions | Compiler Version Security | Pass |
| | | Deprecated Items | Pass |
| | | Redundant Code | Pass |
| | | SafeMath Features | Pass |
| | | require/assert Usage | Pass |
| | | Gas Consumption | Pass |
| | | Visibility Specifiers | Pass |
| | | Fallback Usage | Pass |
| 2 | General Vulnerability | Integer Overflow/Underflow | Pass |
| | | Reentrancy | Pass |
| | | Pseudo-random Number Generator (PRNG) | Pass |
| | | Transaction-Ordering Dependence | Pass |
| | | DoS (Denial of Service) | Pass |
| | | Access Control of Owner | Pass |
| | | Low-level Function (call/delegatecall) Security | Pass |
| | | Returned Value Security | Pass |

| | | | |
|---|---|---|---|
| | | tx.origin Usage | Pass |
| | | Replay Attack | Pass |
| | | Overriding Variables | Pass |
| 3 | Business Security | Business Logics | Pass |
| | | Business Implementations | Pass |

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

## Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts project BLES, including Coding Standards, Security, and Business Logic. **The BLES project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

## Audit Contents:

### 1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

## 2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check the whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

## 3. Business Security

### 3.1 Business analysis of Contract Token BLES

(1) Basic Token Information

| | |
|---|---|
| Token name | Blind Boxes Token |
| Token symbol | BLES |
| decimals | 18 |
| totalSupply | 100 million (Burnable) |
| Token type | ERC20 |

Table 1 Basic Token Information

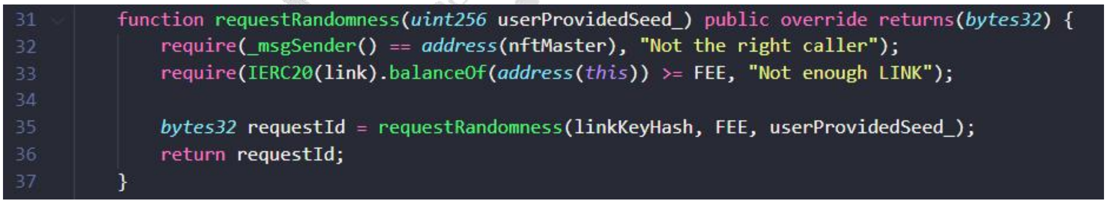
(2) ERC20 Token Standard Functions

- Description: The token contract implements a token which conforms to the ERC20 Standards. It should be noted that the user can directly call the *approve* function to set the approval value for the specified address, but in order to avoid multiple authorizations, it is recommended to use the *increaseAllowance* and *decreaseAllowance* functions when modifying the approval value instead of using the *approve* function directly.
- Related functions: *name, symbol, decimals, totalSupply, balanceOf, allowance, transfer, transferFrom, approve, increaseAllowance, decreaseAllowance, burn, burnFrom*
- Result: Pass

### 3.2 Business analysis of Contract LinkAccessor

(1) Generate random numbers

- Description: The contract implements the *requestRandomness* function for generating random numbers, which requires the caller to be the nftmaster contract, and the balance of link tokens in the contract to be not less than 0.1. After that, the function in the VRFConsumerBase contract of chainlink is called to get the random numbers.



```
    function requestRandomness(uint256 userProvidedSeed_) public override returns(bytes32) {
        require(_msgSender() == address(nftMaster), "Not the right caller");
        require(IERC20(link).balanceOf(address(this)) >= FEE, "Not enough LINK");

        bytes32 requestId = requestRandomness(linkKeyHash, FEE, userProvidedSeed_);
        return requestId;
    }
```

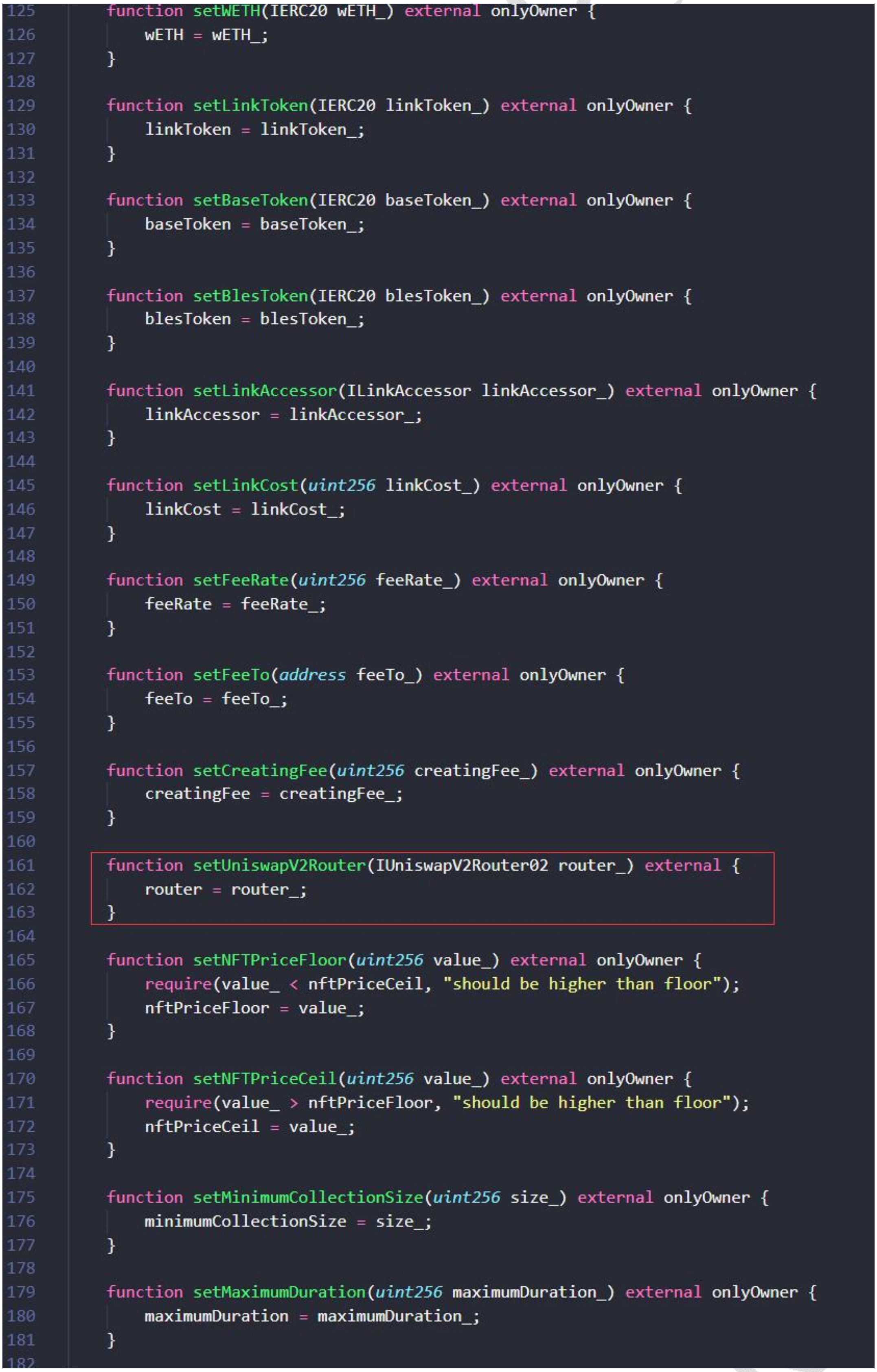
Figure 1 source code of *requestRandomness*

- Related functions: *requestRandomness*
- Result: Pass

### 3.3 Business analysis of Contract NFTMaster

(1) Settings Function

- Description: The contract implements *setWETH*, *setLinkToken*, *setBaseToken*, *setBlesToken*, *setLinkAccessor*, *setLinkCost*, *setFeeRate*, *setFeeTo*, *setCreatingFee*, *setUniswapV2Router*, *setNFTPriceFloor*, *setNFTPriceCeil*, *setMinimumCollectionSize* and *setMaximumDuration* functions are used to modify the contract related parameters, only the owner of the contract can call, the project team declares in the comment that the contract's owner privileges will be transferred to the Timelock contract.

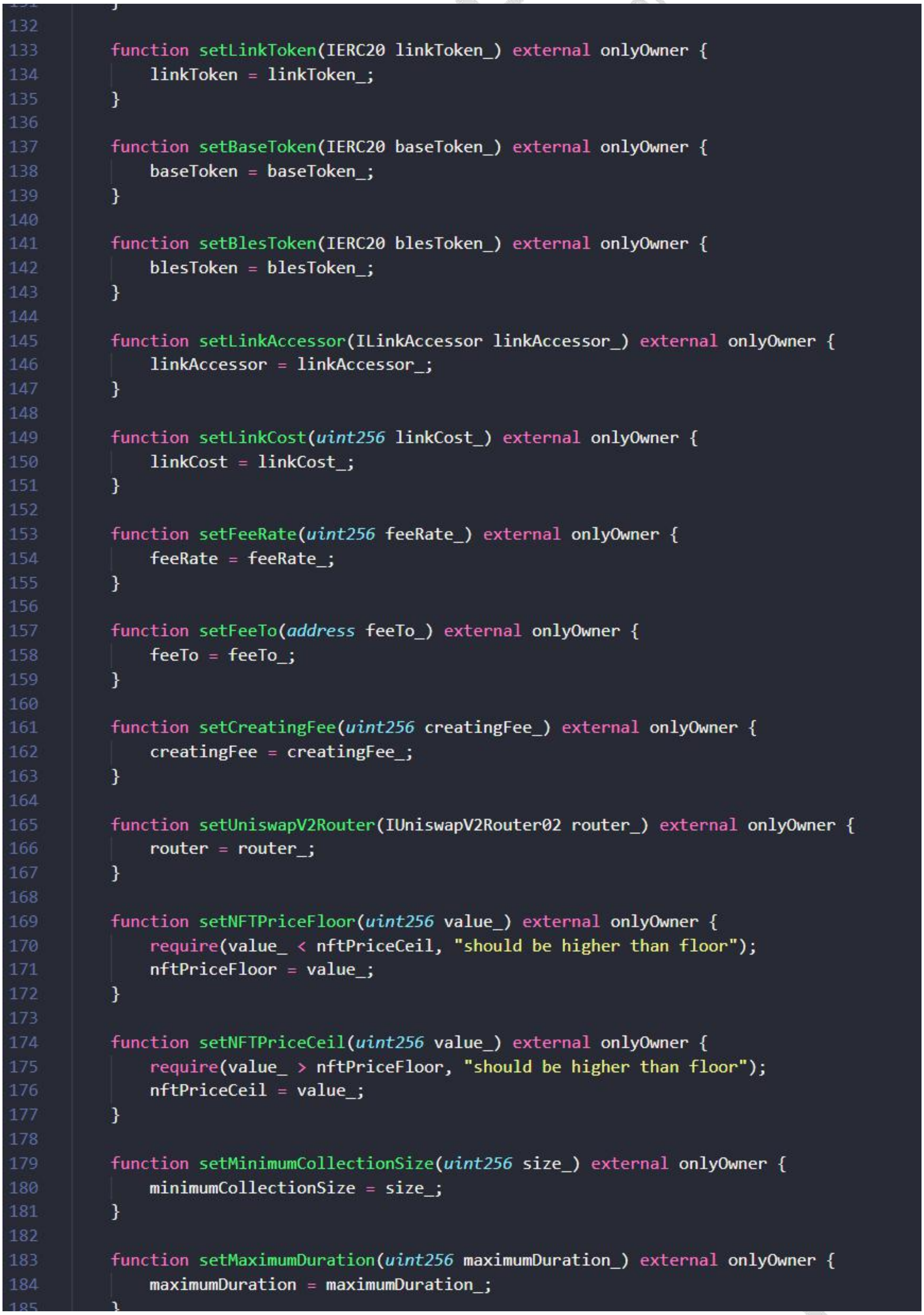


```
    function setWETH(IERC20 wETH_) external onlyOwner {
        wETH = wETH_;
    }

    function setLinkToken(IERC20 linkToken_) external onlyOwner {
        linkToken = linkToken_;
    }

    function setBaseToken(IERC20 baseToken_) external onlyOwner {
        baseToken = baseToken_;
    }

    function setBlesToken(IERC20 blesToken_) external onlyOwner {
        blesToken = blesToken_;
    }

    function setLinkAccessor(ILinkAccessor linkAccessor_) external onlyOwner {
        linkAccessor = linkAccessor_;
    }

    function setLinkCost(uint256 linkCost_) external onlyOwner {
        linkCost = linkCost_;
    }

    function setFeeRate(uint256 feeRate_) external onlyOwner {
        feeRate = feeRate_;
    }

    function setFeeTo(address feeTo_) external onlyOwner {
        feeTo = feeTo_;
    }

    function setCreatingFee(uint256 creatingFee_) external onlyOwner {
        creatingFee = creatingFee_;
    }

    function setUniswapV2Router(IUniswapV2Router02 router_) external {
        router = router_;
    }

    function setNFTPriceFloor(uint256 value_) external onlyOwner {
        require(value_ < nftPriceCeil, "should be higher than floor");
        nftPriceFloor = value_;
    }

    function setNFTPriceCeil(uint256 value_) external onlyOwner {
        require(value_ > nftPriceFloor, "should be higher than floor");
        nftPriceCeil = value_;
    }

    function setMinimumCollectionSize(uint256 size_) external onlyOwner {
        minimumCollectionSize = size_;
    }

    function setMaximumDuration(uint256 maximumDuration_) external onlyOwner {
        maximumDuration = maximumDuration_;
    }

```

Figure 2 source code of related functions (Origin)

- Safety Recommendation: *setUniswapV2Router* function lacks permission judgment, any user can modify, suggest adding *onlyOwner* modifier.

- Repair Result: Fixed



```
    }

    function setLinkToken(IERC20 linkToken_) external onlyOwner {
        linkToken = linkToken_;
    }

    function setBaseToken(IERC20 baseToken_) external onlyOwner {
        baseToken = baseToken_;
    }

    function setBlesToken(IERC20 blesToken_) external onlyOwner {
        blesToken = blesToken_;
    }

    function setLinkAccessor(ILinkAccessor linkAccessor_) external onlyOwner {
        linkAccessor = linkAccessor_;
    }

    function setLinkCost(uint256 linkCost_) external onlyOwner {
        linkCost = linkCost_;
    }

    function setFeeRate(uint256 feeRate_) external onlyOwner {
        feeRate = feeRate_;
    }

    function setFeeTo(address feeTo_) external onlyOwner {
        feeTo = feeTo_;
    }

    function setCreatingFee(uint256 creatingFee_) external onlyOwner {
        creatingFee = creatingFee_;
    }

    function setUniswapV2Router(IUniswapV2Router02 router_) external onlyOwner {
        router = router_;
    }

    function setNFTPriceFloor(uint256 value_) external onlyOwner {
        require(value_ < nftPriceCeil, "should be higher than floor");
        nftPriceFloor = value_;
    }

    function setNFTPriceCeil(uint256 value_) external onlyOwner {
        require(value_ > nftPriceFloor, "should be higher than floor");
        nftPriceCeil = value_;
    }

    function setMinimumCollectionSize(uint256 size_) external onlyOwner {
        minimumCollectionSize = size_;
    }

    function setMaximumDuration(uint256 maximumDuration_) external onlyOwner {
        maximumDuration = maximumDuration_;
    }
```

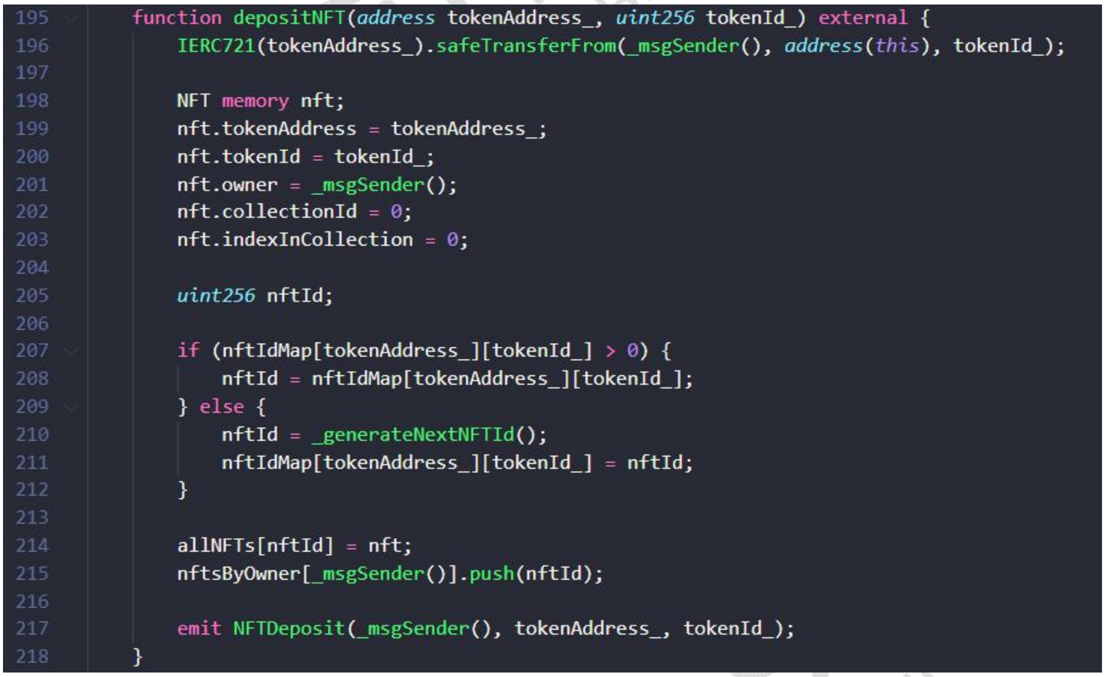
Figure 3 source code of related functions (Fixed)

- Related functions: *setWETH, setLinkToken, setBaseToken, setBlesToken, setLinkAccessor, setLinkCost, setFeeRate, setFeeTo, setCreatingFee, setUniswapV2Router, setNFTPriceFloor, setNFTPriceCeil, setMinimumCollectionSize, setMaximumDuration*

- Result: Pass

(2) Deposit

- Description: The contract implements the *depositNFT* function for staking NFT holdings to this contract after pre-authorization by the user.



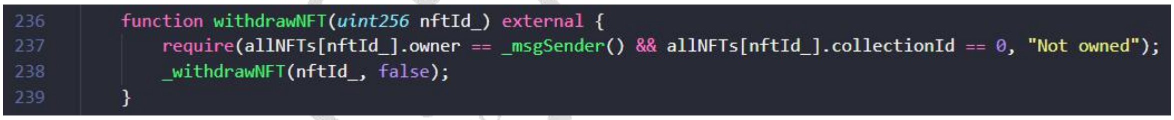
```
    function depositNFT(address tokenAddress_, uint256 tokenId_) external {
        IERC721(tokenAddress_).safeTransferFrom(_msgSender(), address(this), tokenId_);

        NFT memory nft;
        nft.tokenAddress = tokenAddress_;
        nft.tokenId = tokenId_;
        nft.owner = _msgSender();
        nft.collectionId = 0;
        nft.indexInCollection = 0;

        uint256 nftId;

        if (nftIdMap[tokenAddress_][tokenId_] > 0) {
            nftId = nftIdMap[tokenAddress_][tokenId_];
        } else {
            nftId = _generateNextNFTId();
            nftIdMap[tokenAddress_][tokenId_] = nftId;
        }

        allNFTs[nftId] = nft;
        nftsByOwner[_msgSender()].push(nftId);

        emit NFTDeposit(_msgSender(), tokenAddress_, tokenId_);
    }
```

Figure 4 source code of *depositNFT*

- Related functions: *depositNFT*
- Result: Pass

(3) Withdraw

- Description: The contract implements the *withdrawNFT* function for the user to withdraw the NFT tokens in the contract, requiring the caller to be the owner of the NFT and that the NFT is not bound to the collection box.



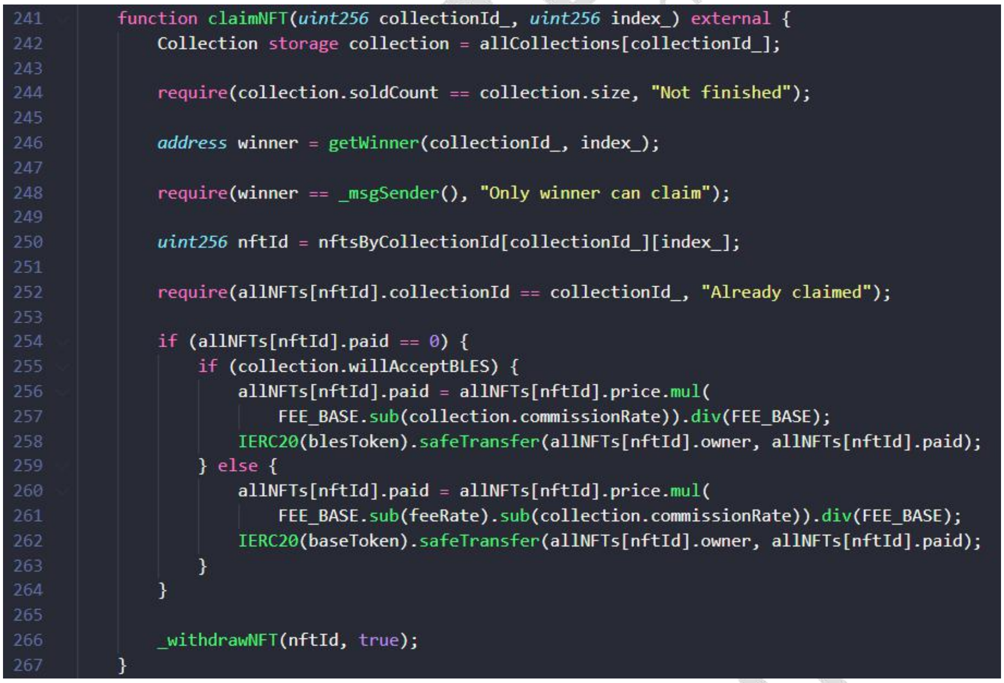
```
    function withdrawNFT(uint256 nftId_) external {
        require(allNFTs[nftId_].owner == _msgSender() && allNFTs[nftId_].collectionId == 0, "Not owned");
        _withdrawNFT(nftId_, false);
    }
```

Figure 5 source code of *withdrawNFT*

- Related functions: *withdrawNFT*
- Result: Pass

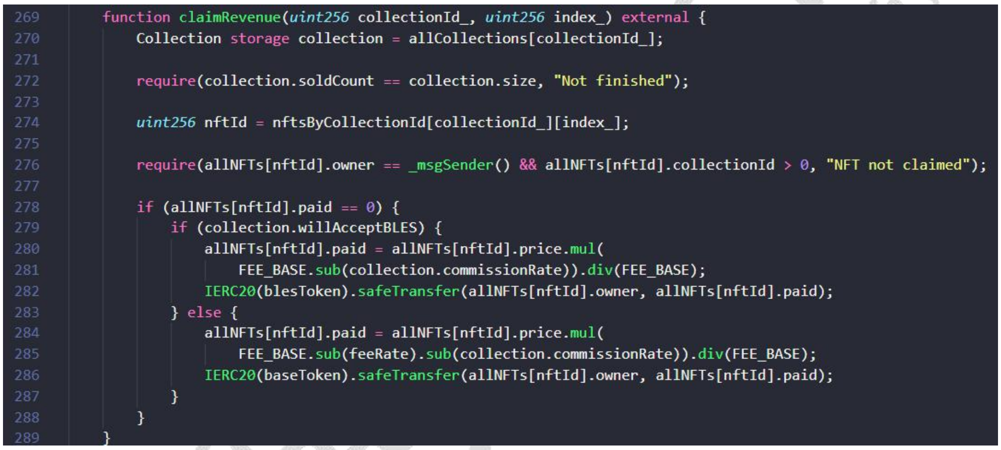
(4) Claim

- Description: The contract implements the *claimNFT* function for the user to claim the NFT obtained by opening the box and to send the reward to the original owner if the original owner of the NFT does not claim the sale reward (BLES or basetoken); *claimRevenue* function is used for the original owner of NFT to collect the sale reward, requiring the call when NFT is not claimed; *claimCommission* function is used for the collection box creator to collect the reward, requiring the call when the collection box is sold out; *claimFee* is used to send basetoken fee to the feeto address, if the collection box uses BLES then no fee will be generated.



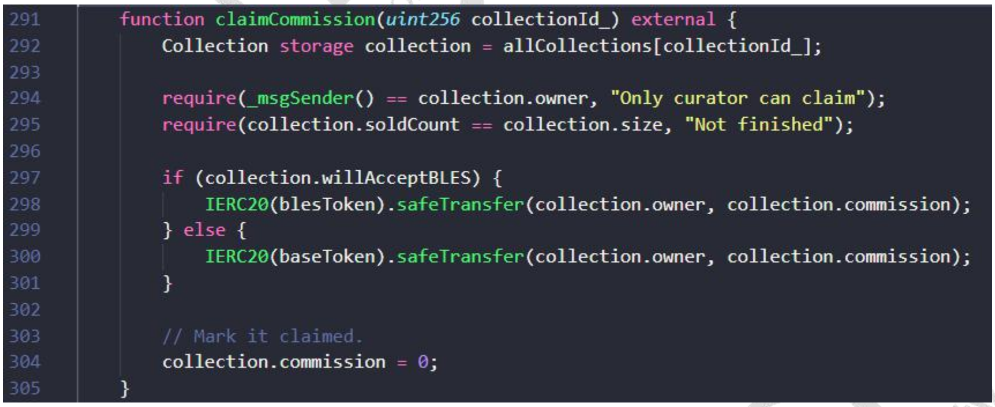
```
    function claimNFT(uint256 collectionId_, uint256 index_) external {
        Collection storage collection = allCollections[collectionId_];

        require(collection.soldCount == collection.size, "Not finished");

        address winner = getWinner(collectionId_, index_);

        require(winner == _msgSender(), "Only winner can claim");

        uint256 nftId = nftsByCollectionId[collectionId_][index_];

        require(allNFTs[nftId].collectionId == collectionId_, "Already claimed");

        if (allNFTs[nftId].paid == 0) {
            if (collection.willAcceptBLES) {
                allNFTs[nftId].paid = allNFTs[nftId].price.mul(
                    FEE_BASE.sub(collection.commissionRate)).div(FEE_BASE);
                IERC20(blesToken).safeTransfer(allNFTs[nftId].owner, allNFTs[nftId].paid);
            } else {
                allNFTs[nftId].paid = allNFTs[nftId].price.mul(
                    FEE_BASE.sub(feeRate).sub(collection.commissionRate)).div(FEE_BASE);
                IERC20(baseToken).safeTransfer(allNFTs[nftId].owner, allNFTs[nftId].paid);
            }
        }

        _withdrawNFT(nftId, true);
    }
```

Figure 6 source code of *claimNFT*



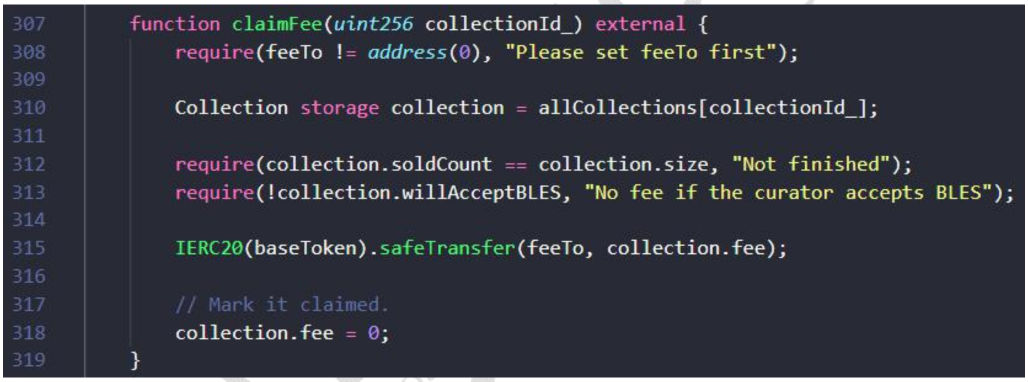
```
    function claimRevenue(uint256 collectionId_, uint256 index_) external {
        Collection storage collection = allCollections[collectionId_];

        require(collection.soldCount == collection.size, "Not finished");

        uint256 nftId = nftsByCollectionId[collectionId_][index_];

        require(allNFTs[nftId].owner == _msgSender() && allNFTs[nftId].collectionId > 0, "NFT not claimed");

        if (allNFTs[nftId].paid == 0) {
            if (collection.willAcceptBLES) {
                allNFTs[nftId].paid = allNFTs[nftId].price.mul(
                    FEE_BASE.sub(collection.commissionRate)).div(FEE_BASE);
                IERC20(blesToken).safeTransfer(allNFTs[nftId].owner, allNFTs[nftId].paid);
            } else {
                allNFTs[nftId].paid = allNFTs[nftId].price.mul(
                    FEE_BASE.sub(feeRate).sub(collection.commissionRate)).div(FEE_BASE);
                IERC20(baseToken).safeTransfer(allNFTs[nftId].owner, allNFTs[nftId].paid);
            }
        }
    }
```

Figure 7 source code of *claimRevenue*



```
    function claimCommission(uint256 collectionId_) external {
        Collection storage collection = allCollections[collectionId_];

        require(_msgSender() == collection.owner, "Only curator can claim");
        require(collection.soldCount == collection.size, "Not finished");

        if (collection.willAcceptBLES) {
            IERC20(blesToken).safeTransfer(collection.owner, collection.commission);
        } else {
            IERC20(baseToken).safeTransfer(collection.owner, collection.commission);
        }

        // Mark it claimed.
        collection.commission = 0;
    }
```

Figure 8 source code of *claimCommission*



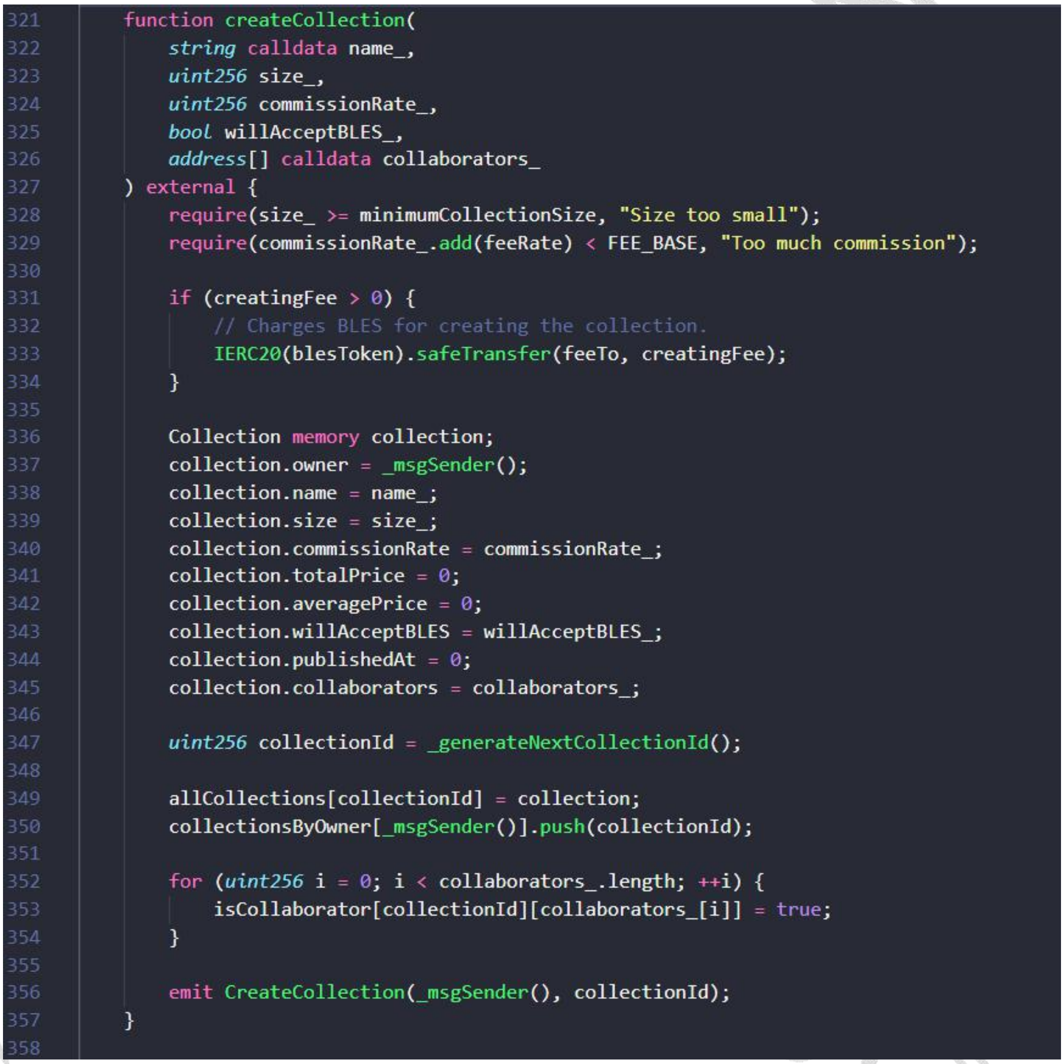
```solidity
function claimFee(uint256 collectionId_) external {
    require(feeTo != address(0), "Please set feeTo first");

    Collection storage collection = allCollections[collectionId_];

    require(collection.soldCount == collection.size, "Not finished");
    require(!collection.willAcceptBLES, "No fee if the curator accepts BLES");

    IERC20(baseToken).safeTransfer(feeTo, collection.fee);

    // Mark it claimed.
    collection.fee = 0;
}
```

Figure 9 source code of *claimFee*

- Related functions: *claimNFT, claimRevenue, claimCommission, claimFee*
- Result: Pass

(5) Create Collection

- Description: The contract implements *createCollection* for the user to create a new collection box. size_ needs to be greater than the minimumCollectionSize, commissionRate_ must not be too high, and depending on the actual situation, creation may require payment of BLES to the feeto address.



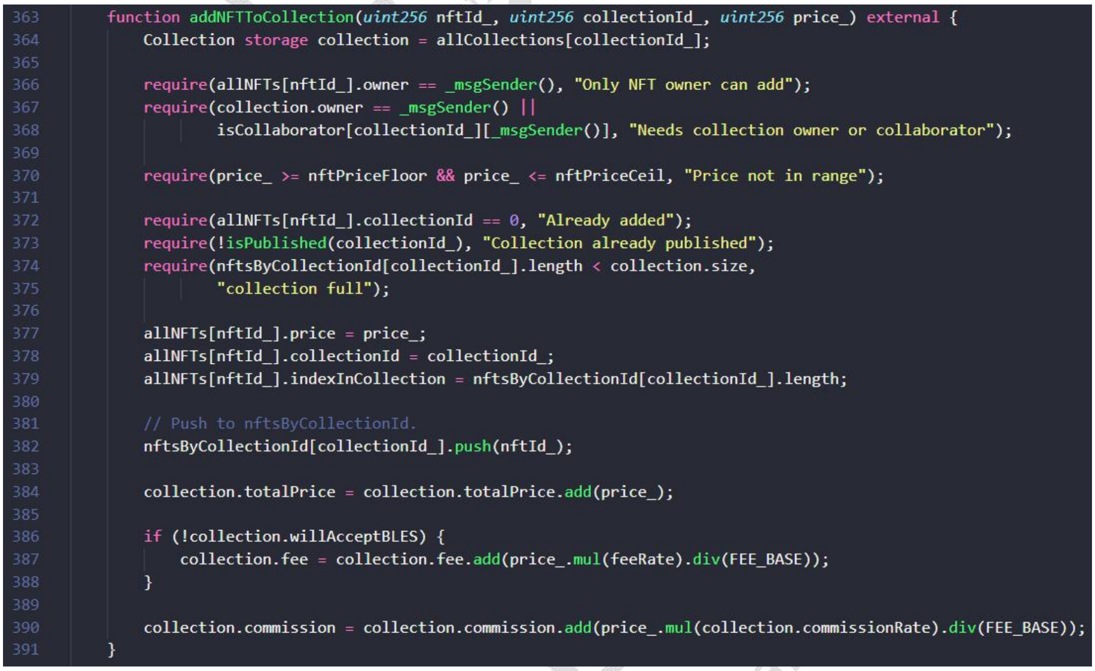
```solidity
function createCollection(
    string calldata name_,
    uint256 size_,
    uint256 commissionRate_,
    bool willAcceptBLES_,
    address[] calldata collaborators_
) external {
    require(size_ >= minimumCollectionSize, "Size too small");
    require(commissionRate_.add(feeRate) < FEE_BASE, "Too much commission");

    if (creatingFee > 0) {
        // Charges BLES for creating the collection.
        IERC20(blesToken).safeTransfer(feeTo, creatingFee);
    }

    Collection memory collection;
    collection.owner = _msgSender();
    collection.name = name_;
    collection.size = size_;
    collection.commissionRate = commissionRate_;
    collection.totalPrice = 0;
    collection.averagePrice = 0;
    collection.willAcceptBLES = willAcceptBLES_;
    collection.publishedAt = 0;
    collection.collaborators = collaborators_;

    uint256 collectionId = _generateNextCollectionId();

    allCollections[collectionId] = collection;
    collectionsByOwner[_msgSender()].push(collectionId);

    for (uint256 i = 0; i < collaborators_.length; ++i) {
        isCollaborator[collectionId][collaborators_[i]] = true;
    }

    emit CreateCollection(_msgSender(), collectionId);
}
```

Figure 10 source code of *createCollection*

- Related functions: *createCollection*
- Result: Pass

(6) Add Collection

- Description: The contract implements *addNFTToCollection* to add the pledged NFT to the collection box, which requires the caller to be the owner of the NFT, have the collection box management rights, set the price to meet the interval, the NFT is not added to other, the collection box is not published The collection box has not reached the limit.



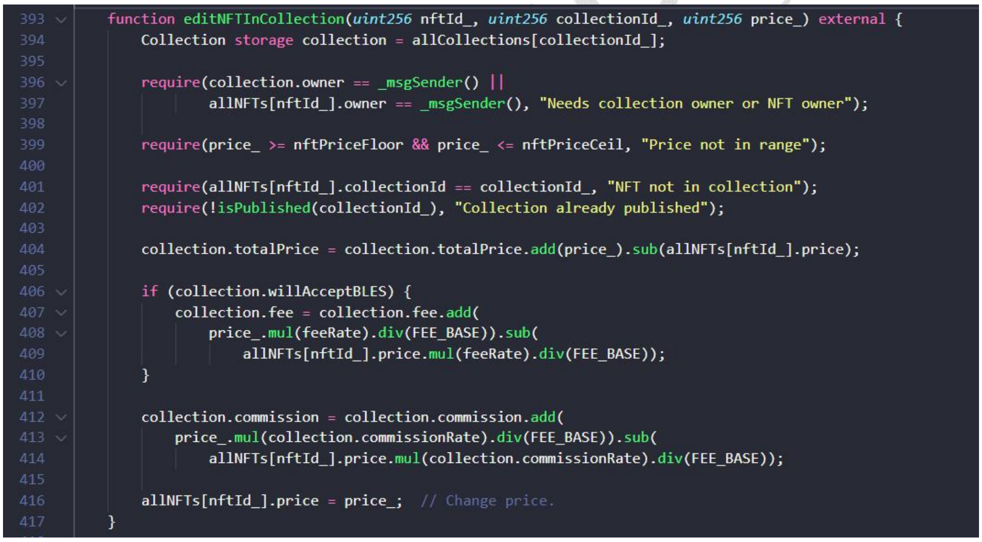
```
    function addNFTToCollection(uint256 nftId_, uint256 collectionId_, uint256 price_) external {
        Collection storage collection = allCollections[collectionId_];

        require(allNFTs[nftId_].owner == _msgSender(), "Only NFT owner can add");
        require(collection.owner == _msgSender() ||
                isCollaborator[collectionId_][_msgSender()], "Needs collection owner or collaborator");

        require(price_ >= nftPriceFloor && price_ <= nftPriceCeil, "Price not in range");

        require(allNFTs[nftId_].collectionId == 0, "Already added");
        require(!isPublished(collectionId_), "Collection already published");
        require(nftsByCollectionId[collectionId_].length < collection.size,
                "collection full");

        allNFTs[nftId_].price = price_;
        allNFTs[nftId_].collectionId = collectionId_;
        allNFTs[nftId_].indexInCollection = nftsByCollectionId[collectionId_].length;

        // Push to nftsByCollectionId.
        nftsByCollectionId[collectionId_].push(nftId_);

        collection.totalPrice = collection.totalPrice.add(price_);

        if (!collection.willAcceptBLES) {
            collection.fee = collection.fee.add(price_.mul(feeRate).div(FEE_BASE));
        }

        collection.commission = collection.commission.add(price_.mul(collection.commissionRate).div(FEE_BASE));
    }
```

Figure 11 source code of *addNFTToCollection*

- Related functions: *addNFTToCollection*
- Result: Pass

(7) Edit Collection

- Description: The contract implements *editNFTInCollection* for modifying the information in the collection box before publishing, requiring the caller to be the owner of the specified NFT or the owner of the collection box.



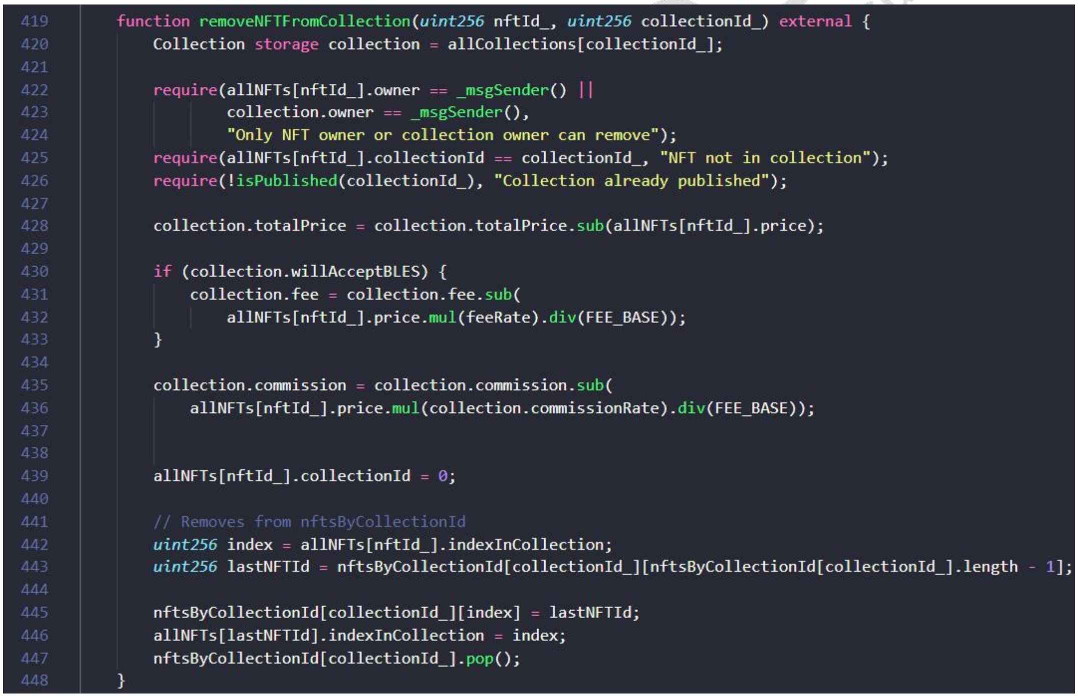
```
function editNFTInCollection(uint256 nftId_, uint256 collectionId_, uint256 price_) external {
    Collection storage collection = allCollections[collectionId_];

    require(collection.owner == _msgSender() ||
            allNFTs[nftId_].owner == _msgSender(), "Needs collection owner or NFT owner");

    require(price_ >= nftPriceFloor && price_ <= nftPriceCeil, "Price not in range");

    require(allNFTs[nftId_].collectionId == collectionId_, "NFT not in collection");
    require(!isPublished(collectionId_), "Collection already published");

    collection.totalPrice = collection.totalPrice.add(price_).sub(allNFTs[nftId_].price);

    if (collection.willAcceptBLES) {
        collection.fee = collection.fee.add(
            price_.mul(feeRate).div(FEE_BASE)).sub(
                allNFTs[nftId_].price.mul(feeRate).div(FEE_BASE));
    }

    collection.commission = collection.commission.add(
        price_.mul(collection.commissionRate).div(FEE_BASE)).sub(
            allNFTs[nftId_].price.mul(collection.commissionRate).div(FEE_BASE));

    allNFTs[nftId_].price = price_;  // Change price.
}
```

Figure 12 source code of *editNFTInCollection*

- Related functions: *editNFTInCollection*
- Result: Pass

(8) Remove Collection

- Description: The contract implements *removeNFTFromCollection* to remove the specified NFT from the collection box before it is published, requiring the caller to be the owner of the specified NFT or the owner of the collection box.



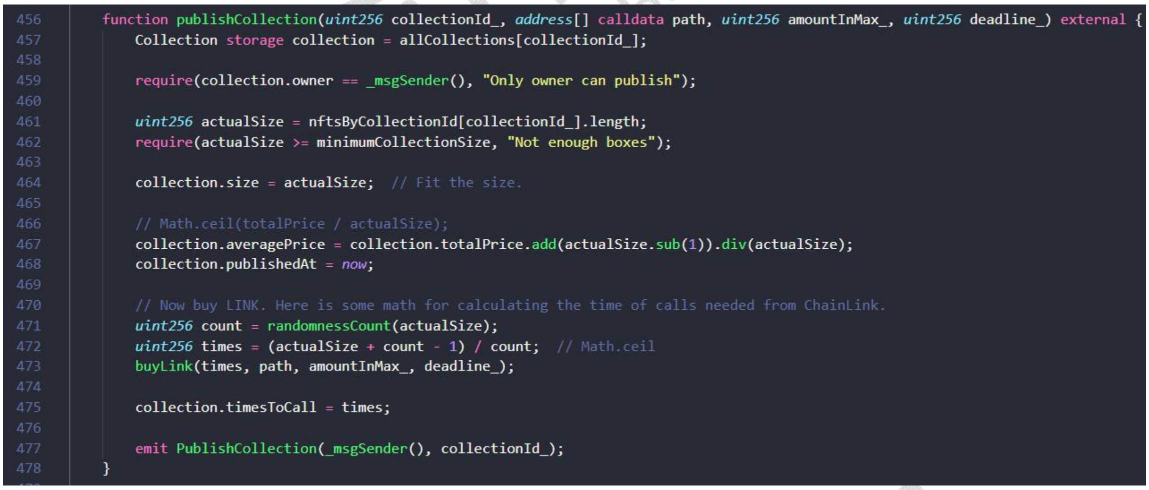
```
function removeNFTFromCollection(uint256 nftId_, uint256 collectionId_) external {
    Collection storage collection = allCollections[collectionId_];

    require(allNFTs[nftId_].owner == _msgSender() ||
            collection.owner == _msgSender(),
            "Only NFT owner or collection owner can remove");
    require(allNFTs[nftId_].collectionId == collectionId_, "NFT not in collection");
    require(!isPublished(collectionId_), "Collection already published");

    collection.totalPrice = collection.totalPrice.sub(allNFTs[nftId_].price);

    if (collection.willAcceptBLES) {
        collection.fee = collection.fee.sub(
            allNFTs[nftId_].price.mul(feeRate).div(FEE_BASE));
    }

    collection.commission = collection.commission.sub(
        allNFTs[nftId_].price.mul(collection.commissionRate).div(FEE_BASE));


    allNFTs[nftId_].collectionId = 0;

    // Removes from nftsByCollectionId
    uint256 index = allNFTs[nftId_].indexInCollection;
    uint256 lastNFTId = nftsByCollectionId[collectionId_][nftsByCollectionId[collectionId_].length - 1];

    nftsByCollectionId[collectionId_][index] = lastNFTId;
    allNFTs[lastNFTId].indexInCollection = index;
    nftsByCollectionId[collectionId_].pop();
}
```

Figure 13 source code of *removeNFTFromCollection*

- Related functions: *removeNFTFromCollection*
- Result: Pass

(9) Publish Collection

- Description: The contract implements *publishCollection* for users to publish their own collection box for other users to purchase. When publishing, the actual number of collection boxes is obtained and the average price of each box is calculated; afterward, the *buyLink* function is called to exchange link tokens to obtain random numbers from the chainlink.



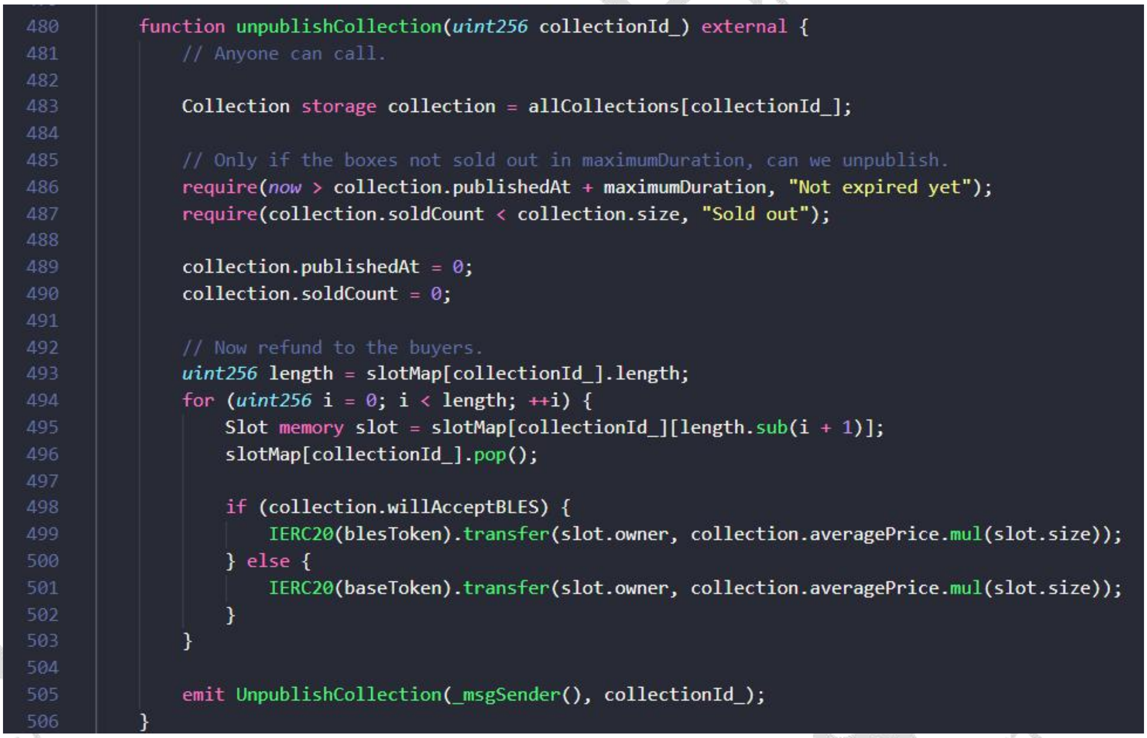
```
    function publishCollection(uint256 collectionId_, address[] calldata path, uint256 amountInMax_, uint256 deadline_) external {
        Collection storage collection = allCollections[collectionId_];

        require(collection.owner == _msgSender(), "Only owner can publish");

        uint256 actualSize = nftsByCollectionId[collectionId_].length;
        require(actualSize >= minimumCollectionSize, "Not enough boxes");

        collection.size = actualSize;  // Fit the size.

        // Math.ceil(totalPrice / actualSize);
        collection.averagePrice = collection.totalPrice.add(actualSize.sub(1)).div(actualSize);
        collection.publishedAt = now;

        // Now buy LINK. Here is some math for calculating the time of calls needed from ChainLink.
        uint256 count = randomnessCount(actualSize);
        uint256 times = (actualSize + count - 1) / count;  // Math.ceil
        buyLink(times, path, amountInMax_, deadline_);

        collection.timesToCall = times;

        emit PublishCollection(_msgSender(), collectionId_);
    }
```

Figure 14 source code of *publishCollection*

- Related functions: *publishCollection*
- Result: Pass

(10) Unpublish Collection

- Description: The contract implements the *unpublishCollection* function for shelving the specified collection box, any user can call, requiring the call time from the publish time is greater than maximumDuration, not sold out. After the refund will be given to users who have purchased.



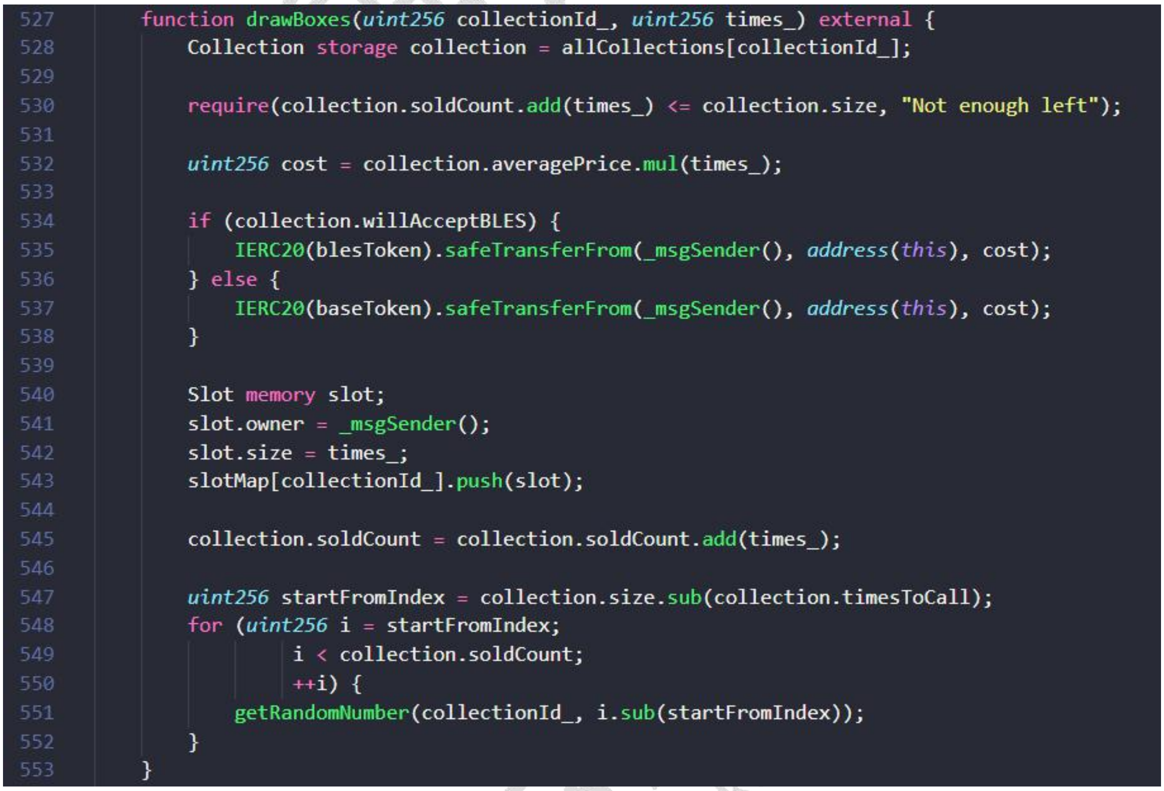
```
    function unpublishCollection(uint256 collectionId_) external {
        // Anyone can call.

        Collection storage collection = allCollections[collectionId_];

        // Only if the boxes not sold out in maximumDuration, can we unpublish.
        require(now > collection.publishedAt + maximumDuration, "Not expired yet");
        require(collection.soldCount < collection.size, "Sold out");

        collection.publishedAt = 0;
        collection.soldCount = 0;

        // Now refund to the buyers.
        uint256 length = slotMap[collectionId_].length;
        for (uint256 i = 0; i < length; ++i) {
            Slot memory slot = slotMap[collectionId_][length.sub(i + 1)];
            slotMap[collectionId_].pop();

            if (collection.willAcceptBLES) {
                IERC20(blesToken).transfer(slot.owner, collection.averagePrice.mul(slot.size));
            } else {
                IERC20(baseToken).transfer(slot.owner, collection.averagePrice.mul(slot.size));
            }
        }

        emit UnpublishCollection(_msgSender(), collectionId_);
    }
```

Figure 15 source code of *unpublishCollection*

- Related functions: *unpublishCollection*
- Result: Pass

(11) Open Box

- Description: The contract implements the *drawBoxes* function for the user to purchase a specified collection box. can be purchased more than one at a time, requiring the number of purchases is not greater than the total number of collection box purchased collection box in the NFT random.



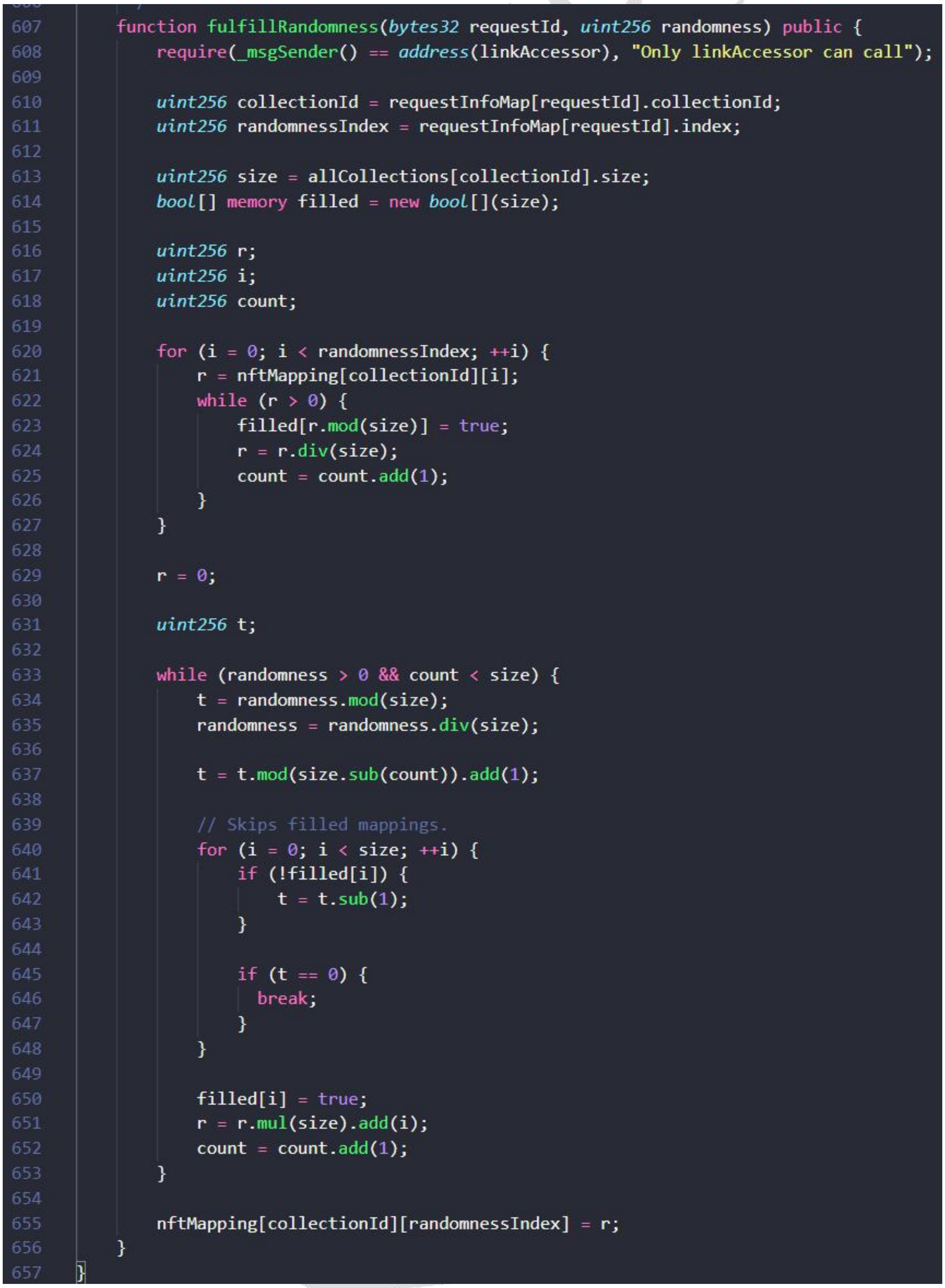
```
function drawBoxes(uint256 collectionId_, uint256 times_) external {
    Collection storage collection = allCollections[collectionId_];

    require(collection.soldCount.add(times_) <= collection.size, "Not enough left");

    uint256 cost = collection.averagePrice.mul(times_);

    if (collection.willAcceptBLES) {
        IERC20(blesToken).safeTransferFrom(_msgSender(), address(this), cost);
    } else {
        IERC20(baseToken).safeTransferFrom(_msgSender(), address(this), cost);
    }

    Slot memory slot;
    slot.owner = _msgSender();
    slot.size = times_;
    slotMap[collectionId_].push(slot);

    collection.soldCount = collection.soldCount.add(times_);

    uint256 startFromIndex = collection.size.sub(collection.timesToCall);
    for (uint256 i = startFromIndex;
            i < collection.soldCount;
            ++i) {
        getRandomNumber(collectionId_, i.sub(startFromIndex));
    }
}
```

Figure 16 source code of *drawBoxes*

- Related functions: *drawBoxes*
- Result: Pass

(12) fulfillRandomness function

- Description: The contract implements the *fullfillRandomness* function as a callback function for the VRF Coordinator contract, only the linkAccessor contract can be called, which will update the relevant data.



```
    function fulfillRandomness(bytes32 requestId, uint256 randomness) public {
        require(_msgSender() == address(linkAccessor), "Only linkAccessor can call");

        uint256 collectionId = requestInfoMap[requestId].collectionId;
        uint256 randomnessIndex = requestInfoMap[requestId].index;

        uint256 size = allCollections[collectionId].size;
        bool[] memory filled = new bool[](size);

        uint256 r;
        uint256 i;
        uint256 count;

        for (i = 0; i < randomnessIndex; ++i) {
            r = nftMapping[collectionId][i];
            while (r > 0) {
                filled[r.mod(size)] = true;
                r = r.div(size);
                count = count.add(1);
            }
        }

        r = 0;

        uint256 t;

        while (randomness > 0 && count < size) {
            t = randomness.mod(size);
            randomness = randomness.div(size);

            t = t.mod(size.sub(count)).add(1);

            // Skips filled mappings.
            for (i = 0; i < size; ++i) {
                if (!filled[i]) {
                    t = t.sub(1);
                }

                if (t == 0) {
                  break;
                }
            }

            filled[i] = true;
            r = r.mul(size).add(i);
            count = count.add(1);
        }

        nftMapping[collectionId][randomnessIndex] = r;
    }
}
```

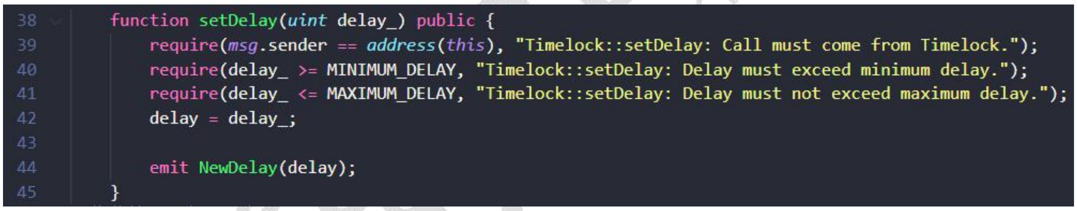
Figure 17 source code of *fulfillRandomness*

- Related functions: *fulfillRandomness*
- Result: Pass

3.4 Business analysis of Contract Timelock

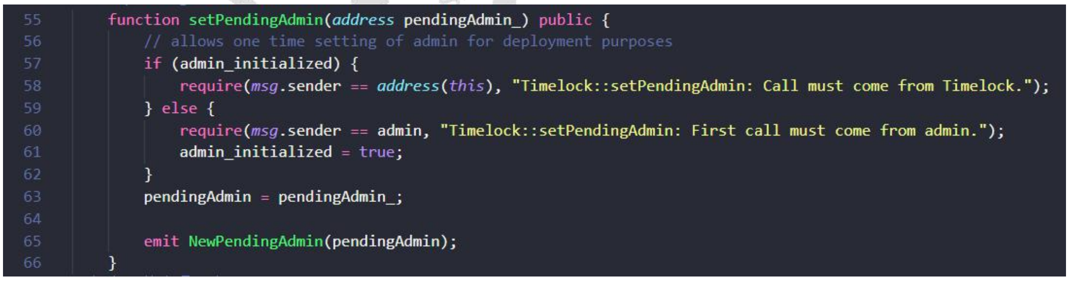
(1) Settings

- Description: The contract implements *setDelay* function for modifying the delay, which can only be called by the contract itself and cannot be modified out of range; *setPendingAdmin* function is used to modify the contract's pendingAdmin, the first time only the administrator can call it, after that only the contract itself can call it.



```solidity
function setDelay(uint delay_) public {
    require(msg.sender == address(this), "Timelock::setDelay: Call must come from Timelock.");
    require(delay_ >= MINIMUM_DELAY, "Timelock::setDelay: Delay must exceed minimum delay.");
    require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum delay.");
    delay = delay_;

    emit NewDelay(delay);
}
```

Figure 18 source code of *setDelay*



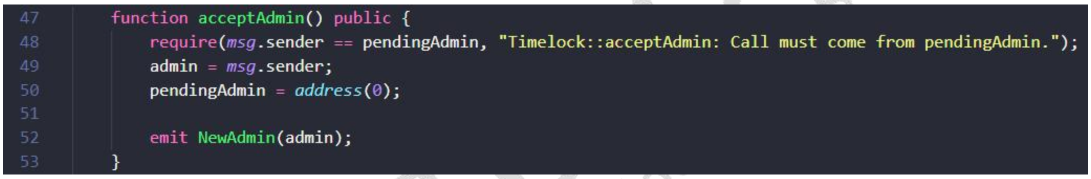
```solidity
function setPendingAdmin(address pendingAdmin_) public {
    // allows one time setting of admin for deployment purposes
    if (admin_initialized) {
        require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from Timelock.");
    } else {
        require(msg.sender == admin, "Timelock::setPendingAdmin: First call must come from admin.");
        admin_initialized = true;
    }
    pendingAdmin = pendingAdmin_;

    emit NewPendingAdmin(pendingAdmin);
}
```

Figure 19 source code of *setPendingAdmin*

- Related functions: *setDelay, setPendingAdmin*
- Result: Pass

(2) Accept owner

- Description: The contract implements the *acceptAdmin* function for the pendingAdmin to receive admin permissions, requiring the caller to be the current pendingAdmin.



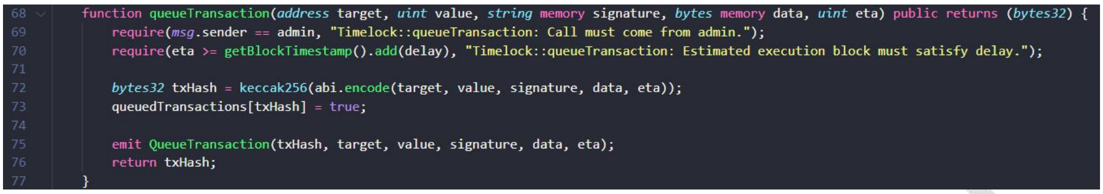
```solidity
function acceptAdmin() public {
    require(msg.sender == pendingAdmin, "Timelock::acceptAdmin: Call must come from pendingAdmin.");
    admin = msg.sender;
    pendingAdmin = address(0);

    emit NewAdmin(admin);
}
```

Figure 20 source code of *acceptAdmin*

- Related functions: *acceptAdmin*
- Result: Pass

(3) Queue

- Description: The contract implements the *queueTransaction* function for submitting a transaction, available only to admin, eta needs to be no less than delay from the current time.



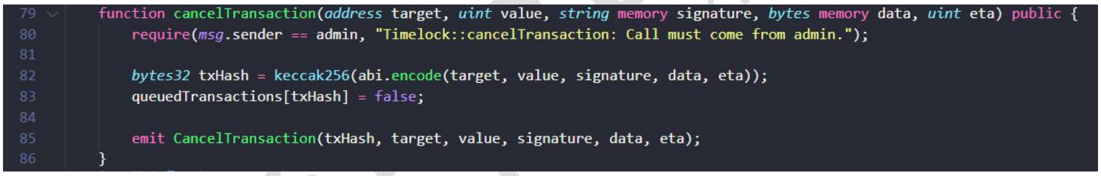
```solidity
function queueTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public returns (bytes32) {
    require(msg.sender == admin, "Timelock::queueTransaction: Call must come from admin.");
    require(eta >= getBlockTimestamp().add(delay), "Timelock::queueTransaction: Estimated execution block must satisfy delay.");

    bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
    queuedTransactions[txHash] = true;

    emit QueueTransaction(txHash, target, value, signature, data, eta);
    return txHash;
}
```

Figure 21 source code of *queueTransaction*

- Related functions: *queueTransaction*
- Result: Pass

(4) Cancel

- Description: The contract implements the *cancelTransaction* function for cancelling a transaction, available only to admin.



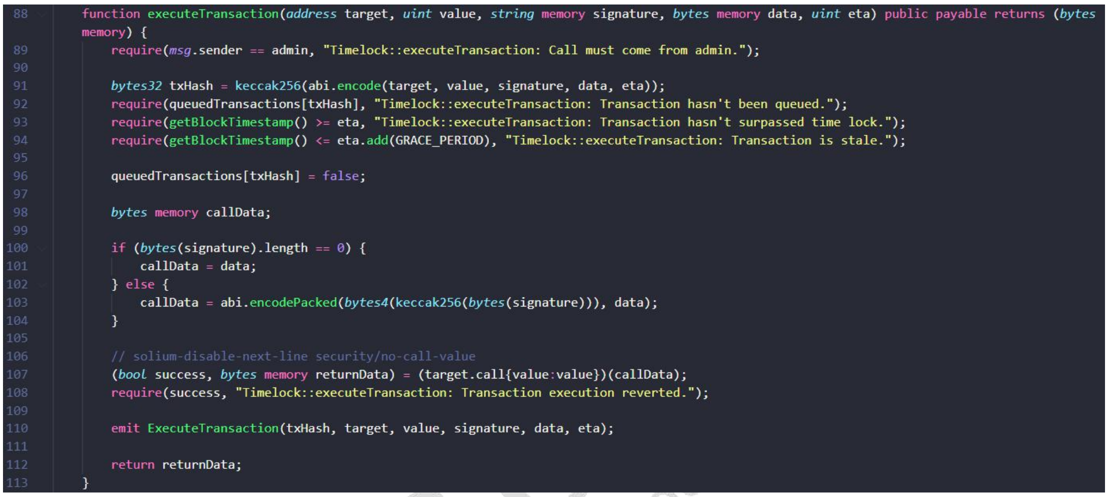
```
    function cancelTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public {
        require(msg.sender == admin, "Timelock::cancelTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = false;

        emit CancelTransaction(txHash, target, value, signature, data, eta);
    }
```

Figure 22 source code of *cancelTransaction*

- Related functions: *cancelTransaction*
- Result: Pass

(5) Execute

- Description: The contract implements the *executeTransaction* function for executing a transaction, available only to admin. Requires that the transaction has been submitted and that the current time is greater than eta but does not exceed eta time GRACE_PERIOD.



```
    function executeTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public payable returns (bytes
    memory) {
        require(msg.sender == admin, "Timelock::executeTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        require(queuedTransactions[txHash], "Timelock::executeTransaction: Transaction hasn't been queued.");
        require(getBlockTimestamp() >= eta, "Timelock::executeTransaction: Transaction hasn't surpassed time lock.");
        require(getBlockTimestamp() <= eta.add(GRACE_PERIOD), "Timelock::executeTransaction: Transaction is stale.");

        queuedTransactions[txHash] = false;

        bytes memory callData;

        if (bytes(signature).length == 0) {
            callData = data;
        } else {
            callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
        }

        // solium-disable-next-line security/no-call-value
        (bool success, bytes memory returnData) = (target.call{value:value})(callData);
        require(success, "Timelock::executeTransaction: Transaction execution reverted.");

        emit ExecuteTransaction(txHash, target, value, signature, data, eta);

        return returnData;
    }
```

Figure 23 source code of *executeTransaction*

- Related functions: *executeTransaction*
- Result: Pass

## 4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project BLES. The problems found by the audit team during the audit process have been notified to the project party and reached an agreement on the repair results, the overall audit result of the BLES project's smart contract is **Pass**.

Beosin

BEOSIN

Blockchain Security

**Official Website**

https://lianantech.com

**E-mail**

vaas@lianantech.com

**Twitter**

https://twitter.com/Beosin_com