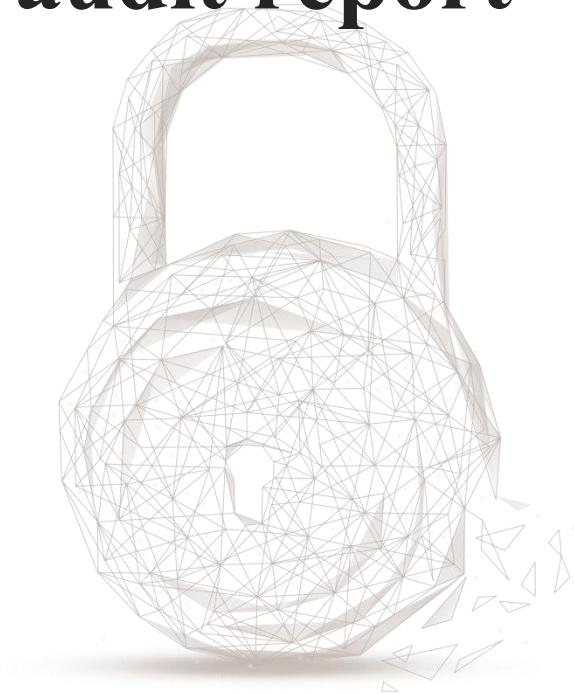
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202104021425

Report Query Name: JGNNFT

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
JGNNFT1155	0x3E31F70912c00AEa971A8b2045bd568D738C31Dc	https://bscscan.com/address/0x3E31F70912c00AEa971A8b2045bd568D738C31Dc#code
JGNNFT721	0x3e855B7941fE8ef5F07DAd68C5140D6a3EC1b286	https://bscscan.com/address/0x3e855B7941fE8ef5F07DAd68C5140D6a3EC1b286#code

Start Date：2021.03.30

Completion Date：2021.04.02

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 JGNNFT, including Coding Standards, Security, and Business Logic. **The JGNNFT 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 BNB.
- 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 JGNNFT1155

(1) Basic Token Information

Token name	JGNNFT
Token symbol	JGNNFT
contractURI	https://jgnnft.com/api/nft1155
tokenURIPrefix	https://jgnnft.com/api/1155/
Token type	BEP-1155

Table 1 Basic Token Information

(2) mint function

- Description: The contract implements the *mint* function for the user to mint new tokens. When minting, you need to set the token ID, fee related data, total number of tokens and URI, and will check if the token ID already exists and if the fee receiving address is 0 address. Any user can call this function to create their own NFT.

```
1024    function mint(uint256 id, Fee[] memory fees, uint256 supply, string memory uri) public {
1025        _mint(id, fees, supply, uri);
1026    }
```

Figure 1 source code of *mint*

```
864     // Creates a new token type and assings _initialSupply to minter
865     function _mint(uint256 _id, Fee[] memory _fees, uint256 _supply, string memory _uri) internal {
866         require(creators[_id] == address(0x0), "Token is already minted");
867         require(_supply != 0, "Supply should be positive");
868         require(bytes(_uri).length > 0, "uri should be set");
869
870         creators[_id] = msg.sender;
871         address[] memory recipients = new address[](_fees.length);
872         uint[] memory bps = new uint[](_fees.length);
873         for (uint i = 0; i < _fees.length; i++) {
874             require(_fees[i].recipient != address(0x0), "Recipient should be present");
875             require(_fees[i].value != 0, "Fee value should be positive");
876             fees[_id].push(_fees[i]);
877             recipients[i] = _fees[i].recipient;
878             bps[i] = _fees[i].value;
879         }
880         if (_fees.length > 0) {
881             emit SecondarySaleFees(_id, recipients, bps);
882         }
883         balances[_id][msg.sender] = _supply;
884         _setTokenURI(_id, _uri);
885
886         // Transfer event with mint semantic
887         emit TransferSingle(msg.sender, address(0x0), msg.sender, _id, _supply);
888         emit URI(_uri, _id);
889     }
```

Figure 2 source code of *_mint*

- Related functions: *mint*
- Result: Pass

(3) Transfer function

- Description: The contract implements *safeBatchTransferFrom* and *safeTransferFrom* functions for proxy transferring and sending after the user is authorized. The *safeBatchTransferFrom* function can be

called to transfer multiple token id tokens at once, while *safeTransferFrom* function can only transfer one type at a time. If the transfer target address is a contract, the relevant function is called to perform a check to determine whether the target address can accept BEP-1155 tokens.

```
431     function safeBatchTransferFrom(address _from, address _to, uint256[] calldata _ids, uint256[] calldata _values, bytes calldata _data) external {
432
433         // MUST Throw on errors
434         require(_to != address(0x0), "destination address must be non-zero.");
435         require(_ids.length == _values.length, "_ids and _values array lenght must match.");
436         require(_from == msg.sender || operatorApproval[_from][msg.sender] == true, "Need operator approval for 3rd party transfers.");
437
438         for (uint256 i = 0; i < _ids.length; ++i) {
439             uint256 id = _ids[i];
440             uint256 value = _values[i];
441
442             // SafeMath will throw with insuficient funds _from
443             // or if _id is not valid (balance will be 0)
444             balances[id][_from] = balances[id][_from].sub(value);
445             balances[id][_to]   = value.add(balances[id][_to]);
446         }
447
448         // Note: instead of the below batch versions of event and acceptance check you MAY have emitted a TransferSingle
449         // event and a subsequent call to _doSafeTransferAcceptanceCheck in above loop for each balance change instead.
450         // Or emitted a TransferSingle event for each in the loop and then the single _doSafeBatchTransferAcceptanceCheck below.
451         // However it is implemented the balance changes and events MUST match when a check (i.e. calling an external contract) is done.
452
453         // MUST emit event
454         emit TransferBatch(msg.sender, _from, _to, _ids, _values);
455
456         // Now that the balances are updated and the events are emitted,
457         // call onERC1155BatchReceived if the destination is a contract.
458         if (_to.isContract()) {
459             _doSafeBatchTransferAcceptanceCheck(msg.sender, _from, _to, _ids, _values, _data);
460         }
461     }
```

Figure 3 source code of *safeBatchTransferFrom*

```
395     function safeTransferFrom(address _from, address _to, uint256 _id, uint256 _value, bytes calldata _data) external {
396
397         require(_to != address(0x0), "_to must be non-zero.");
398         require(_from == msg.sender || operatorApproval[_from][msg.sender] == true, "Need operator approval for 3rd party transfers.");
399
400         // SafeMath will throw with insuficient funds _from
401         // or if _id is not valid (balance will be 0)
402         balances[_id][_from] = balances[_id][_from].sub(_value);
403         balances[_id][_to]   = _value.add(balances[_id][_to]);
404
405         // MUST emit event
406         emit TransferSingle(msg.sender, _from, _to, _id, _value);
407
408         // Now that the balance is updated and the event was emitted,
409         // call onERC1155Received if the destination is a contract.
410         if (_to.isContract()) {
411             _doSafeTransferAcceptanceCheck(msg.sender, _from, _to, _id, _value, _data);
412         }
413     }
```

Figure 4 source code of *safeTransferFrom*

- Related functions: *safeBatchTransferFrom, safeTransferFrom*
- Result: Pass

(4) Burn function

- Description: The contract implements the *burn* function to destroy a specified amount of tokens of a specified token id of _owner address.

```
891     function burn(address _owner, uint256 _id, uint256 _value) external {
892
893         require(_owner == msg.sender || operatorApproval[_owner][msg.sender] == true, "Need operator approval for 3rd party burns.");
894
895         // SafeMath will throw with insuficient funds _owner
896         // or if _id is not valid (balance will be 0)
897         balances[_id][_owner] = balances[_id][_owner].sub(_value);
898
899         // MUST emit event
900         emit TransferSingle(msg.sender, _owner, address(0x0), _id, _value);
901     }
```

Figure 5 source code of *burn*

- Related functions: *burn*
- Result: Pass

(5) Approve function

- Description: The contract implements the *setApprovalForAll* function for authorizing to the specified address. Note: This authorization is of type bool, unlike the uint type of *approve* function in the BEP-20 standard, and after authorization, all of the user's BEP-1155 token can be manipulated (all types of token held by the user, no upper limit on the amount). This function can be called again to cancel the authorization.

```
502     function setApprovalForAll(address _operator, bool _approved) external {
503         operatorApproval[msg.sender][_operator] = _approved;
504         emit ApprovalForAll(msg.sender, _operator, _approved);
505     }
```

Figure 6 source code of *setApprovalForAll*

- Related functions: *setApprovalForAll*
- Result: Pass

(6) Signer function

- Description: The contract implements the *addSigner* function for granting Signer permission, the *removeSigner* function for removing user Signer permission, and *renounceSigner* function for the caller to renounce user's Signer permission. Note: In the contract, the Signer permission has no practical effect, and the related functions are redundant code, so it is recommended to delete them.

```
1015    function addSigner(address account) public onlyOwner {
1016        _addSigner(account);
1017    }
1018
1019    function removeSigner(address account) public onlyOwner {
1020        _removeSigner(account);
1021    }
```

Figure 7 source code of *addSigner* and *removeSigner*

```
983     function renounceSigner() public {
984         _removeSigner(_msgSender());
985     }
```

Figure 8 source code of *renounceSigner*

- Related functions: *addSigner, removeSigner, renounceSigner*
- Security Advice: The related functions are redundant code, so it is recommended to delete them.
- Repair Result: Ignore, does not affect the normal use of the contract.
- Result: Pass

(7) Other functions

- Description: The contract implements *setTokenURIPrefix* function and *setContractURI* function for modifying contract-related information, which can only be called by the contract owner.

```
914     function setTokenURIPrefix(string memory tokenURIPrefix) public onlyOwner {
915         _setTokenURIPrefix(tokenURIPrefix);
916     }
917
918     function setContractURI(string memory contractURI) public onlyOwner {
919         _setContractURI(contractURI);
920     }
921 }
```

Figure 9 source code of *setTokenURIPrefix* and *setContractURI*

- Related functions: *setTokenURIPrefix*, *setContractURI*
- Result: Pass

3.2 Business analysis of Contract Token JGNNFT721

(1) Basic Token Information

Token name	JGNNFT
Token symbol	JGNNFT
contractURI	https://jgnnft.com/api/nft721
tokenURIPrefix	https://jgnnft.com/api/721/
Token type	BEP-721

Table 2 Basic Token Information

(2) mint function

- Description: The contract implements the *mint* function for the user to mint new tokens. When minting, you need to set the token ID, fee related data and URI, and will check if the token ID already exists and if the fee receiving address is 0 address. Any user can call this function to create their own NFT.

```
1387    function mint(uint256 tokenId, Fee[] memory _fees, string memory tokenURI) public {
1388        _mint(msg.sender, tokenId, _fees);
1389        _setTokenURI(tokenId, tokenURI);
1390    }
```

Figure 10 source code of *mint*

```
1320    function _mint(address to, uint256 tokenId, Fee[] memory _fees) internal {
1321        _mint(to, tokenId);
1322        address[] memory recipients = new address[](_fees.length);
1323        uint[] memory bps = new uint[](_fees.length);
1324        for (uint i = 0; i < _fees.length; i++) {
1325            require(_fees[i].recipient != address(0x0), "Recipient should be present");
1326            require(_fees[i].value != 0, "Fee value should be positive");
1327            fees[tokenId].push(_fees[i]);
1328            recipients[i] = _fees[i].recipient;
1329            bps[i] = _fees[i].value;
1330        }
1331        if (_fees.length > 0) {
1332            emit SecondarySaleFees(tokenId, recipients, bps);
1333        }
1334    }
```

Figure 11 source code of *_mint*

- Related functions: *mint*
- Result: Pass

(3) Transfer function

- Description: The contract implements two *safeTransferFrom* and one *transferFrom* functions for proxy transfers when the user is authorized. The difference is that users can use the *transferFrom* function to transfer their tokens directly; one of the *safeTransferFrom* functions supports sending data along with the transfer. Transferring requires entering the from address, to address and token id.

```
688     function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data) public {
689         require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");
690         _safeTransferFrom(from, to, tokenId, _data);
691     }
```

Figure 12 source code of *safeTransferFrom*(with data)

```
672     function safeTransferFrom(address from, address to, uint256 tokenId) public {
673         safeTransferFrom(from, to, tokenId, "");
674     }
```

Figure 13 source code of *safeTransferFrom*

```
654     function transferFrom(address from, address to, uint256 tokenId) public {
655         //solhint-disable-next-line max-line-length
656         require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");
657
658         _transferFrom(from, to, tokenId);
659     }
```

Figure 14 source code of *transferFrom*

- Related functions: *safeTransferFrom, transferFrom*
- Result: Pass

(4) Burn function

- Description: The contract implements the *burn* function to destroy the specified id tokens, requiring the caller to have authorization or be the owner of the tokens.

```
888     function burn(uint256 tokenId) public {
889         //solhint-disable-next-line max-line-length
890         require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721Burnable: caller is not owner nor approved");
891         _burn(tokenId);
892     }
```

Figure 15 source code of *burn*

- Related functions: *burn*
- Result: Pass

(5) Approve function

- Description: The contract implements the *approve* function and *setApprovalForAll* function for authorization. *approve* function is used to authorize tokens with specified token id to the specified address. *setApprovalForAll* function is used to give permission to the specified address to operate all tokens of the owner.

```
599     function approve(address to, uint256 tokenId) public {
600         address owner = ownerOf(tokenId);
601         require(to != owner, "ERC721: approval to current owner");
602 
603         require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
604             "ERC721: approve caller is not owner nor approved for all"
605         );
606 
607         _tokenApprovals[tokenId] = to;
608         emit Approval(owner, to, tokenId);
609     }
```

Figure 16 source code of *approve*

- Related functions: *approve, setApprovalForAll*
- Result: Pass

(6) Other functions

- Description: The contract implements *setTokenURIPrefix* and *setContractURI* for modifying contract-related information, which can only be called by the contract owner.

```
1392     function setTokenURIPrefix(string memory tokenURIPrefix) public onlyOwner {
1393         _setTokenURIPrefix(tokenURIPrefix);
1394     }
1395 
1396     function setContractURI(string memory contractURI) public onlyOwner {
1397         _setContractURI(contractURI);
1398     }
```

Figure 17 source code of *setTokenURIPrefix* and *setContractURI*

- Related functions: *setTokenURIPrefix*, *setContractURI*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project JGNNFT. The problems found by the audit team during the audit process have been notified to the project party and agree on the outcome of the restoration. The overall audit result of the JGNNFT project's smart contract is **Pass**.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com