QuillAudits

Audit Report
June, 2023

For

DonBangSeok

Table of Content

Executive Summary

Project Name	Donbangseok
Overview	The contract is an ERC721 token contract that integrates Openzeppelin library; . The utilities aid in tracking token id counters, URI management, and access control mechanism. The contract allows the contract owner to mint a token to a recipient address while at the same time assigning receivers for the token royalties.
Timeline	29th May, 2023 to 30th May, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Language	Solidity
Blockchain	Polygon
Scope of Audit	The scope of this audit was to analyze the Donbangseok codebase for quality, security, and correctness. *https://gitlab.com/icraft-bdt-ngin/nft-royalty-contract/-/blob/main/contracts/DBSNft.sol* Branch: Main Commit: 13da102165e7b66fc322eda72b7c90b4c82372e6
Contracts in Scope	DBSNft.sol
Fixed in	77a80c96d3b455cf207debe9ad07985d2c1772bd

High Medium

Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	**2**	**2**

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - DBSNft.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A.1 Missing Zero Address Check

Description

Some functions in the contract require providing an array of addresses to perform its functionality, however, it fails to check the inclusivity of a null address among the parameters. While it is certain that when the mintNFT function is called, it will revert when null address is passed into it due to the present of the null address check prevent in the safeMint internal function called from ERC721 inherited from Openzeppelin, there is a need to add the check to the updateRoyalties function to avoid setting a null address as a receiver of the null address.

Remediation

Add a null address check to the updateRoyalties to prevent making a null address as a receiver.

Status

Resolved

https://gitlab.com/icraft-bdt-ngin/nft-royalty-contract/-/commit/11c1263ae528880259b8a240bce2b1b9aba652ea

A.2 Transfer of Ownership Must be a Two-way process

Description

Contracts are integrated with the standard Openzeppelin ownable contract, however, when the owner mistakenly transfers ownership to an incorrect address, ownership is completely removed from the original owner and cannot be reverted. The transferOwnership() function in the ownable contract allows the current owner to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored in the storage, after validating the newOwner is a non-zero address, which may not be enough.

Remediation

It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership() . Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner , instead of modifying the _owner() directly. The updateOwnership() function checks whether _newOwner is msg.sender, which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner. You can also use the Openzeppelin Ownable2.

Status

Resolved

https://gitlab.com/icraft-bdt-ngin/nft-royalty-contract/-/commit/785747c81ba69a87d18b235bce26307561701903

Informational Issues

A.3 Floating Solidity Version (pragma solidity ^0.8.11)

Description

Contract has a floating solidity pragma version. This is present also in inherited contracts. Locking the pragma helps to ensure that the contract does not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. The recent solidity pragma version also possesses its own unique bugs.

Remediation

Making the contract use a stable solidity pragma version prevents bugs occurrence that could be ushered in by prospective versions. It is recommended, therefore, to use a fixed solidity pragma version while deploying to avoid deployment with versions that could expose the contract to attack.

Status

Resolved

https://gitlab.com/icraft-bdt-ngin/nft-royalty-contract/-/commit/666a0d443a21155a5eb004bd074f6cf830a7624a

A.4 Variable Overshadowing

Description

At the constructor level, the name and symbol variables overshadow the similar variables names used in the original ERC721 contract inherited. This could cause possible breaks in the contract control flow.

Remediation

It is recommended to rename the variables or add an underscore to them to differentiate the variables.

Reference

Status

Resolved

https://gitlab.com/icraft-bdt-ngin/nft-royalty-contract/-/commit/666a0d443a21155a5eb004bd074f6cf830a7624a

Functional Tests

Some of the tests performed are mentioned below

- Should get the token details after the contract is deployed
- Should allow only owner to mint successfully to recipients, alongside the royalties receivers
- Should revert when the number of receivers in an array is unequal to the percentages
- Should revert when address zero is passed into the safeMint function
- Should revert when the total percentage provided for the receivers is not equal to 100
- Should update the royalties of previously minted tokens.
- Should revert when the receiver array has a null address included.
- Should split incoming funds according to identified shares in percentages.
- Should get royalty information from overridden function from ERC 2981.
- Should create a new instance of payment splitter contract on new token mint.
- Should get royalty percentage for the whole ERc 721 tokens.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) ignores return value by IERC721Recei
ver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-447)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
DBSNft.constructor(string,string).name (contracts/DBSNft.sol#26) shadows:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81) (function)
        - IERC721Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#16) (function)
DBSNft.constructor(string,string).symbol (contracts/DBSNft.sol#26) shadows:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88) (function)
        - IERC721Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#21) (function)
DBSNft.mintNFT(address,string,address[],uint256[]).tokenURI (contracts/DBSNft.sol#38) shadows:
        - ERC721URIStorage.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol#20-36) (function)
        - ERC721.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#93-98) (function)
        - IERC721Metadata.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#26) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in DBSNft.mintNFT(address,string,address[],uint256[]) (contracts/DBSNft.sol#36-59):
        External calls:
        - _safeMint(recipient,tokenId) (contracts/DBSNft.sol#51)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-447)
        State variables written after the call(s):
        - _royalties[tokenId] = RoyaltyInfo(receivers,percentages) (contracts/DBSNft.sol#54)
        - _setTokenURI(tokenId,tokenURI) (contracts/DBSNft.sol#52)
                - _tokenURIs[tokenId] = _tokenURI (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol#47)
```


```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) ignores return value by IERC721Recei
ver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-447)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
DBSNft.constructor(string,string).name (contracts/DBSNft.sol#26) shadows:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81) (function)
        - IERC721Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#16) (function)
DBSNft.constructor(string,string).symbol (contracts/DBSNft.sol#26) shadows:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88) (function)
        - IERC721Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#21) (function)
DBSNft.mintNFT(address,string,address[],uint256[]).tokenURI (contracts/DBSNft.sol#38) shadows:
        - ERC721URIStorage.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol#20-36) (function)
        - ERC721.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#93-98) (function)
        - IERC721Metadata.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#26) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in DBSNft.mintNFT(address,string,address[],uint256[]) (contracts/DBSNft.sol#36-59):
        External calls:
        - _safeMint(recipient,tokenId) (contracts/DBSNft.sol#51)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-447)
        State variables written after the call(s):
        - _royalties[tokenId] = RoyaltyInfo(receivers,percentages) (contracts/DBSNft.sol#54)
        - _setTokenURI(tokenId,tokenURI) (contracts/DBSNft.sol#52)
                - _tokenURIs[tokenId] = _tokenURI (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol#47)
```


```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#155-162):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#160)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#180-187):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function ERC721.__unsafe_increaseBalance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#503-505) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:. analyzed (14 contracts with 85 detectors), 39 result(s) found
```

Summary

In this report, we have considered the security of Donbangseok. We performed our audit according to the procedure described above.

Some issues of low and informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Donbangseok Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Donbangseok Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$16B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2023

For

DonBangSeok

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com