QuillAudits

Audit Report
July, 2022

For

Table of Content

Table of Content

Executive Summary

Project Name	Jadu
Overview	Jadu is combining magic & technology to build the definitive Web3 Augmented Reality platform.
Timeline	19 August,2022 to 26 August,2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Jadu smart contract codebase for quality, security, and correctness. *https://github.com/JaduAR/ava-contracts* **Commit hash:** 4e141c06efa81d21dd264c0614ff77c2c267c896
Fixed In	1fbb1c0a31950f6b77fdad02fbcfab4261ff35ba

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	3	0	1	3

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 - JaduAVA.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

A1. Inheriting from unused contract

Description

The JaduAVA token contract inherits the Ownable contract but none of its functionality is used. All the required logic is already present in the AccessControl contract

Remediation

We recommend removing Ownable contract from JaduAVA.sol

Status

Fixed

B. Contract - JaduAVADistributor.sol

High Severity Issues

B1. Mint size passed by the user is ignored

Line	Code

117-133	function mintNft(
	address to,
	uint256 size,
	uint256 nonce,
	bytes calldata signature,
	MintLimitInfo[] calldata mintLimitInfos
	) external payable returns (bool) {
	uint256 sizeToMint = _mintChecks(
	to,
	size,
	nonce,
	signature,
	mintLimitInfos
	); // Verifying the transaction data and parameters
	jaduAVAToken.mint(_msgSender(), sizeToMint); // Minting the tokens
	return true;
	}

178-190	function _mintChecks(
	address to,
	uint256 size,
	uint256 nonce,
	bytes calldata signature,
	MintLimitInfo[] calldata mintLimitInfos
	) internal returns (uint256) {
	
	....
	
	uint256 wantToMintCount = verifyAndGetMintCount(mintLimitInfos);
	isValidSignature[signature] = true; // setting signature as redeemed

	payable(JADU_TREASURY_WALLET).transfer(msg.value); // transfering amount to treasury wallet
	return wantToMintCount; // returning verified Limit for the transaction .

Description

The user input size is the amount of tokens the user wants to mint. The mint fee and all other logical calculations are also based on this parameter. But the returned value is wantToMintCount which is totally different from the size input passed by the user. Thus the actual minted token amount is equal to wantToMintCount.

Remediation

We recommend using the amount 'size' for minting

Status

Fixed

B2. Token Hash can be manipulated to bypass mint limits

Line	Code
199-204	bytes32 tokenHash = keccak256(abi.encodePacked(mintLimitInfos[index].tokenId, mintLimitInfos[index].contractAddress)); // hash of tokenID and contract address
205	MintLimitInfo memory currentTokenLimit = mintlimitInfo[tokenHash];

Description

mintLimitInfos array is an input passed by the user. The tokenId and contract address parameters are used to generate a tokenHash which is further used to fetch token limits from the contract storage.
Any user can easily pass incorrect values for tokenId and contractAddress to fetch token limits of another tokenId or contract address and bypass the mint limits.

Remediation

We recommend fetching the tokenId and contract address from the token contract instead of taking a user input.

Status

Fixed

The team clarified that the mintLimitInfo will be passed and signed by a Signer, thus tokenHash cannot be tampered with.

B3. Fetch maxLimit from contract storage instead of user input

Line	Code
206-211	require(currentTokenLimit.totalMinted + mintLimitInfos[index].totalMinted <= mintLimitInfos[index].maxLimit, "JADU AVA: Invalid minting limit");

Description

The maxLimit should be fetched from the contract storage instead of using the user input.

Status

Fixed

The team clarified that the mintLimitInfo will be passed and signed by a Signer, thus tokenHash cannot be tampered with.

Medium Severity Issues

B4. Centralization Risk

Description

There are ADMIN and SIGNER roles which have authority to bypass the contract logic and execute several functions. The SIGNER role has the authority to :

- Sign and pass any data for minting
- airdrop tokens

Remediation

The Signer role can sign any invalid data which can be used to mint tokens to any user. So proper care should be taken while signing the data.
We advise the client to handle these accounts carefully to avoid any potential hack. We also advise the client to consider the following solutions:

- Time-lock with reasonable latency for community awareness on privileged operations;
- Multisig with community-voted 3rd-party independent co-signers;
- DAO or Governance module increasing transparency and community involvement;

Status

Acknowledged

Low Severity Issues

B5. Admin can mint tokens even when minting is paused

Description

There are 3 different functions which allow minting of the NFT token. There is a functionality to pause minting as well. When it's paused, no token mint function should work. But there are 2 function which allow admin to mint even when the minting is paused :

- airDropJaduAVA
- jaduVaultAirdrop

Remediation

We recommend checking for _mintingPaused value before minting tokens.

Status

Fixed

Informational Issues

B6. Use calldata instead of memory to save gas

Line	Code

178-190	function verifySignature(
	address wallet,
	uint256 _size,
	uint256 _nonce,
	bytes32 _mintData,
	bytes memory _signature
	) internal pure returns (address) {
	return
	ECDSA.recover(
	keccak256(abi.encode(wallet, _size, _nonce, _mintData)), // recovering SIGNER WALLET
	_signature

Description

Using calldata here will save gas.

Remediation

We recommend replacing memory with calldata.

Status

Fixed

B7. No need to pass wallet parameter for signature verification

Line	Code
83-84	address wallet = _msgSender(); require(wallet == to, "caller and wallet mismatch");
87-93	address signerOwner = verifySignature(wallet, size, nonce, mintData, signature); // Verifying signature

Description

_mintChecks function accepts a parameter ‘to’. This parameter is required to be equal to msg.sender. And then it is also passed to the verifySignature function. If the ‘wallet’ value is equal to msg.sender then it will automatically verify as the Signature will be valid, otherwise it will fail. So you do not need the parameter ‘to’, just directly passing msg.sender to verifySignature will work.

Remediation

We recommend removing the input parameter ‘to’ and directly use msg.sender in verifySignature logic.

Status

Fixed

B8. Redundant max supply check

Line	Code
100-103	require(tokenId + size <= jaduAVAToken.getMaxSupply(), "JADU: Maximum cap reached"); // Max supply check

Description

There is require condition which checks if the current mint amount does not exceed the maximum supply of the token. But such a check is already there in the token contract and this check is not required

Remediation

We recommend removing this max supply check.

Status

Acknowledged

B9. Unused Custom Error

Line	Code
10	error ContractUpgraded();

Description

There is an unused custom error in the contract.

Remediation

We recommend removing the error code or using it in the logic.

Status

Fixed

Functional Tests

A. Contract - JaduAVA.sol

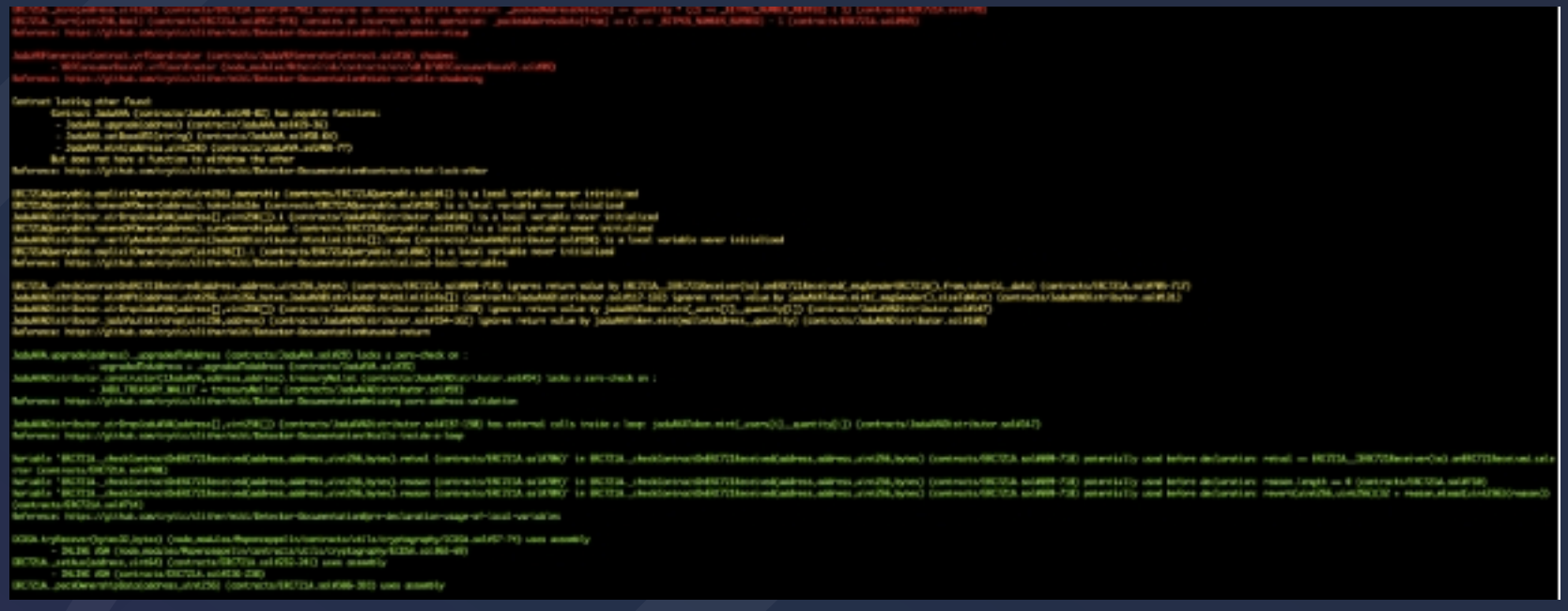
- ✓ Minting of NFTs should increase the totalSupply
- ✓ List of tokens should be correct for every wallet

B. Contract - JaduAVADistributor.sol

- ✓ Minting of NFTs should increase the totalSupply
- ✓ Should not be able to mint more than 10 at a time
- ✓ Should pay MINT FEE to mint

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.



```
ERC721A._setOwnersExplicit(uint256,uint256) (contracts/ERC721A.sol#967-997) uses assembly
        - INLINE ASM (contracts/ERC721A.sol#990-994)
ERC721A._toString(uint256) (contracts/ERC721A.sol#1849-1881) uses assembly
        - INLINE ASM (contracts/ERC721A.sol#1865-1868)
ERC721AQueryable.tokensOfOwnerIn(address,uint256,uint256) (contracts/ERC721AQueryable.sol#83-104) uses assembly
        - INLINE ASM (contracts/ERC721AQueryable.sol#101-102)
console._sendLogPayload(bytes) (node_modules/hardhat/console.sol#7-14) uses assembly
        - INLINE ASM (node_modules/hardhat/console.sol#10-12)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Different versions of Solidity is used:
        - Version used: ['^0.8.0', '^0.8.4']
        - ^0.8.4 (node_modules/@rmrk-team/evm-contracts/src/v0.8/RMRKCatalogImpl.sol#2)
        - ^0.8.0 (node_modules/@rmrk-team/evm-contracts/src/v0.8/interfaces/IRMRKCatalogImpl.sol#2)
        - ^0.8.4 (contracts/JadaNFTGenerationContract.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Different versions of Solidity is used:
        - Version used: ['>=0.4.22<0.9.0', '^0.8.0', '^0.8.4', '^0.8.7']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/AccessControl.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/IAccessControl.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.4 (contracts/ERC721A.sol#4)
        - ^0.8.4 (contracts/ERC721AQueryable.sol#4)
        - ^0.8.4 (contracts/IJadaNFA.sol#2)
        - ^0.8.7 (contracts/JadaNFA.sol#2)
        - ^0.8.7 (contracts/JadaNFAErc/router.sol#2)
        - >=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

ERC721A._mint(address,uint256,bytes,bool) (contracts/ERC721A.sol#299-325) is never used and should be removed
ERC721A._burn(uint256) (contracts/ERC721A.sol#488-500) is never used and should be removed
ERC721A._burn(uint256,bool) (contracts/ERC721A.sol#502-526) is never used and should be removed
ERC721A._getAux(address) (contracts/ERC721A.sol#224-226) is never used and should be removed
```

```
ERC721A._initializeOwnershipAt(uint256) (contracts/ERC721A.sol#930-948) is never used and should be removed
ERC721A._packedOwnershipOf(uint256) (contracts/ERC721A.sol#895-897) is never used and should be removed
ERC721A._numberBurned(address) (contracts/ERC721A.sol#217-219) is never used and should be removed
ERC721A._numberMinted(address) (contracts/ERC721A.sol#210-212) is never used and should be removed
ERC721A._ownershipAt(address,uint256) (contracts/ERC721A.sol#907-909) is never used and should be removed
ERC721A._ownershipOf(address,uint256,bytes) (contracts/ERC721A.sol#882-884) is never used and should be removed
ERC721A._setAux(address,uint64) (contracts/ERC721A.sol#232-242) is never used and should be removed
ERC721A._setExtraDataAt(uint256,uint24) (contracts/ERC721A.sol#957-959) is never used and should be removed
ERC721A._totalBurned() (contracts/ERC721A.sol#191-193) is never used and should be removed
ERC721A._totalMinted() (contracts/ERC721A.sol#180-182) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@rmrk-team/evm-contracts/src/v0.8/interfaces/IRMRKCatalogImpl.sol#2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/AccessControl.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/IAccessControl.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version>=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

IJadaNFA (contracts/IJadaNFA.sol#6-62) should inherit from IJadaNFA (contracts/IJadaNFA.sol#64-68)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance

Variable JadaNFAGenerationContract.CONSOLEADDR (contracts/JadaNFAGenerationContract.sol#8) is not in mixedCase
Variable JadaNFAGenerationContract.s_subscriptionId (contracts/JadaNFAGenerationContract.sol#32) is not in mixedCase
Variable JadaNFAGenerationContract.s_randomWord (contracts/JadaNFAGenerationContract.sol#38) is not in mixedCase
Variable JadaNFAGenerationContract.s_requestId (contracts/JadaNFAGenerationContract.sol#36) is not in mixedCase
Variable JadaNFAGenerationContract.s_owner (contracts/JadaNFAGenerationContract.sol#40) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Contract ERC721A_IERC721Receiver (contracts/ERC721A.sol#3-18) is not in CapWords
Parameter ERC721A.safeTransferFrom(address,address,uint256,bytes)._data (contracts/ERC721A.sol#644) is not in mixedCase
Parameter JadaNFA.upgrade(address)._upgradedAddress (contracts/JadaNFA.sol#25) is not in mixedCase
Variable JadaNFA.MAX_SUPPLY (contracts/JadaNFA.sol#9) is not in mixedCase
Variable JadaNFA._baseTokenURI (contracts/JadaNFA.sol#11) is not in mixedCase
Parameter JadaNFAErc/router.setJadaTreasuryWallet(address)._jadaTreasuryWallet (contracts/JadaNFAErc/router.sol#44) is not in mixedCase
Parameter JadaNFAErc/router.airDropJadaNFA(address[],uint256[])._users (contracts/JadaNFAErc/router.sol#58) is not in mixedCase
Parameter JadaNFAErc/router.airDropJadaNFA(address[],uint256[])._quantity (contracts/JadaNFAErc/router.sol#59) is not in mixedCase
Parameter JadaNFAErc/router.jadaMint(uint256,address,uint256,bytes32,bytes)._quantity (contracts/JadaNFAErc/router.sol#124) is not in mixedCase
Parameter JadaNFAErc/router.toggleMintPause(bool)._pause (contracts/JadaNFAErc/router.sol#58) is not in mixedCase
Parameter JadaNFAErc/router.updateMintLimit(uint256)._newMint (contracts/JadaNFAErc/router.sol#172) is not in mixedCase
Parameter JadaNFAErc/router.verify(signature(address,uint256,uint256,bytes32,bytes32)._sig (contracts/JadaNFAErc/router.sol#108) is not in mixedCase
Parameter JadaNFAErc/router.verify(signature(address,uint256,uint256,bytes32,bytes32)._owner (contracts/JadaNFAErc/router.sol#111) is not in mixedCase
Parameter JadaNFAErc/router.verify(signature(address,uint256,uint256,bytes32,bytes32)._airdrop (contracts/JadaNFAErc/router.sol#112) is not in mixedCase
Parameter JadaNFAErc/router.verify(signature(address,uint256,uint256,bytes32,bytes32)._signature (contracts/JadaNFAErc/router.sol#113) is not in mixedCase
Variable JadaNFAErc/router.MAX_TREASURY_AMOUNT (contracts/JadaNFAErc/router.sol#23) is not in mixedCase
Variable JadaNFAErc/router.MINT_PRICE (contracts/JadaNFAErc/router.sol#22) is not in mixedCase
```

```
Variable JadaNFAErc/router._allStagePaused (contracts/JadaNFAErc/router.sol#27) is not in mixedCase
Contract console (node_modules/hardhat/console.sol#4-1552) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

JadaNFAGenerationContract.slitherConstructorVariables() (contracts/JadaNFAGenerationContract.sol#8-72) uses literals with too many digits:
        - callbackGasLimit = 500000 (contracts/JadaNFAGenerationContract.sol#29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

console.slitherConstructorConstantVariables() (node_modules/hardhat/console.sol#4-1552) uses literals with too many digits:
        - CONSOLE_ADDRESS = address(0x000000000000000000636f6e736f6c652e6c6f67) (node_modules/hardhat/console.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

JadaNFAGenerationContract.callbackGasLimit (contracts/JadaNFAGenerationContract.sol#29) should be constant
JadaNFAGenerationContract.keyHash (contracts/JadaNFAGenerationContract.sol#28) should be constant
JadaNFAGenerationContract.numWords (contracts/JadaNFAGenerationContract.sol#34) should be constant
JadaNFAGenerationContract.requestConfirmations (contracts/JadaNFAGenerationContract.sol#33) should be constant
JadaNFAGenerationContract.vrfCoordinator (contracts/JadaNFAGenerationContract.sol#26) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

JadaNFA.MAX_SUPPLY (contracts/JadaNFA.sol#9) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

randomSeed() should be declared external:
        - JadaNFAGenerationContract.randomSeed() (contracts/JadaNFAGenerationContract.sol#72-74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#144-148)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#158-162)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#175-183)
renounceRemainTo() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-63)
transferRemainTo(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#69-72)
name() should be declared external:
        - ERC721A.name() (contracts/ERC721A.sol#271-273)
symbol() should be declared external:
        - ERC721A.symbol() (contracts/ERC721A.sol#280-282)
tokenURI(uint256) should be declared external:
        - ERC721A.tokenURI(uint256) (contracts/ERC721A.sol#287-292)
approve(address,uint256) should be declared external:
        - ERC721A.approve(address,uint256) (contracts/ERC721A.sol#429-434)
getApproved(uint256) should be declared external:
        - ERC721A.getApproved(uint256) (contracts/ERC721A.sol#441-447)
setApprovalForAll(address,bool) should be declared external:
        - ERC721A.setApprovalForAll(address,bool) (contracts/ERC721A.sol#449-455)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721A.safeTransferFrom(address,address,uint256) (contracts/ERC721A.sol#457-463)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (20 contracts with 77 detectors), 106 result(s) found
```

Closing Summary

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

Some issues of High, Medium, 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 Jadu 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 Jadu 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.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2022

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com