QuillAudits

Audit Report
October, 2022

For

ARTSWAP

Table of Content

Table of Content

Table of Content

Executive Summary

Project Name	ArtSwap NFT Contract
Overview	In ArtSwap NFT contract users can create their own galleries to display the NFTs, mint NFTs, airdrop and transfer them to other users. In the Marketplace contract these NFT available to buy and sell. Also resale, secondarySell, addGallery functionalities are available.
Timeline	14/09/2022 to 23/09/2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze ArtSwap NFT Contract codebase for quality, security, and correctness. *https://github.com/artswap-rumsan/artswap-contracts* Commit Hash: 08b5bbde71a73b87666b87ceb29ca83599508c26
Fixed In	*https://github.com/artswap-rumsan/artswap-contracts/tree/11-random-string* Commit Hash: f49086b52983f606d800fdedfb99e46d4fe0a3a9

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	1	1	2	9

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 - Gallery

High Severity Issues

A.1 Bypassing the check to steal the NFT in verifyAirDrop()

Line	Function - verifyAirDrop()
417 - 432	

```
function verifyAirDrop(
    address _to,
    uint256 _tokenId,
    uint256 _randomnumber
) public {
    AirDropInfo storage airdrop = airDropInfo[_tokenId];
    bytes32 _code = getHash(_randomnumber);
    require(airdrop.verificationCode == _code, 'Invalid Code');
    require(listOfTokenIds.contains(_tokenId), 'N/A in this gallery');
    if (tokeninfo[_tokenId].onSell) cancelNftSell(_tokenId);
    airdrop.isClaimed = true;
    airdrop.receiver = _to;
    address owner = nft.ownerOf(_tokenId);
    nft.safeTransferFrom(owner, _to, _tokenId);
    emit NftAirdropped(_tokenId, _to);
}
```

Description

According to the intended behavior the user receives the verification code via email. After that he can verify and the NFT will be transferred to him/her. But as only a random number (not truly random) is used to verify the process it is quite possible that a malicious user can guess the number from the available public mapping of airDropInfo and call the verifyAirDrop() for himself and claim the NFT because the _to address is set after the verification code check

Remediation

For the mitigation if user is already known to which NFT is being airdropped then make sure to set the value in mintandAirDropwithVerfication()
As airdrop.receiver = _to and add check in verifyAirDrop as require(airdrop.receiver == _to, “Invalid user”); or can add system for whitelisting users

Status

Resolved

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

A.2: Possibility of msg.sender/user getting set twice for Gallery

Line	Function - constructor() in Gallery contract
39-54	

```
constructor(
    string memory _id,
    address _owner, //gallery owner address
    address _nft,
    address _market // address _dollarmarket
) {
    id = _id;
    creator = _owner;
    nft = INFT(_nft);
    admins[_owner] = true;
    admins[msg.sender] = true;
    market = IMarketPlace(_market);
    transferOwnership(_owner);
    blockNumber = block.number;
    // market_dollar = IMarketPlace(_dollarmarket);
}
```

Description

From GalleryFactory contract when gallery is getting created user will create gallery for himself. So here in the constructor admins[_owner] and admins[msg.sender] can be set as msg.sender only.

Remediation

it is advisable to set admins once only to not get duplicate if possible.

Status

Resolved

Update: The client dev team said that the owner and msg.sender are not the same user so we need to add both users as admin

A.3: Unlocked pragma (pragma solidity ^0.8.1)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same.

Status

Resolved

A.4: Misleading comment in code

Description

In tranferNft() and manageAirDropWithVerification() the comments are misleading.

Remediation

In transferNft() , token id to burn should be token id to transfer
In manageAirDropWithVerification(), airfrop should be aidrop

Status

Resolved

A.5 General Recommendation

Description

buyNft() in Gallery contract is payable function. Here it is not immune to re-entrancy attack. But to be safe and according best practices it is advisable to use Re-entrancyguard.

Use similar naming structure
mapping(uint256 => TokenInfo) public tokeninfo;
mapping(uint256 => feeInfo) public FeeInfo;
mapping(uint256 => AirDropInfo) public airDropInfo;
As you can see tokenInfo, airDropInfo mappings are small letter first alphabet and FeeInfo is capital letter first. Please follow the same naming structure.

function mintandAirDropwithVerfication()has spelling mistakes please make sure to fix it.

Status

Resolved

B. Contract - NFT

High Severity Issues

No issues found

Medium Severity Issues

B.1 Anyone can set royalties for NFTs in setArtistRoyalty()

Line	Function - setArtistRoyalty()

67-73	`function setArtistRoyalty(`
	`  uint256 _tokenId,`
	`  address _receiver,`
	`  uint96 _feeNumerator`
	`) public override {`
	`  _setTokenRoyalty(_tokenId, _receiver, _feeNumerator);`
	`}`

Description

Function setArtistRoyalty() has public visibility means anyone can call it. For some reason if NFT is minted through NFT contract and not from gallery then it is quite possible that it can be call by anyone(malicious user) to set the royalty amount to malicious user address which can send the royalty amount to he's address

Remediation

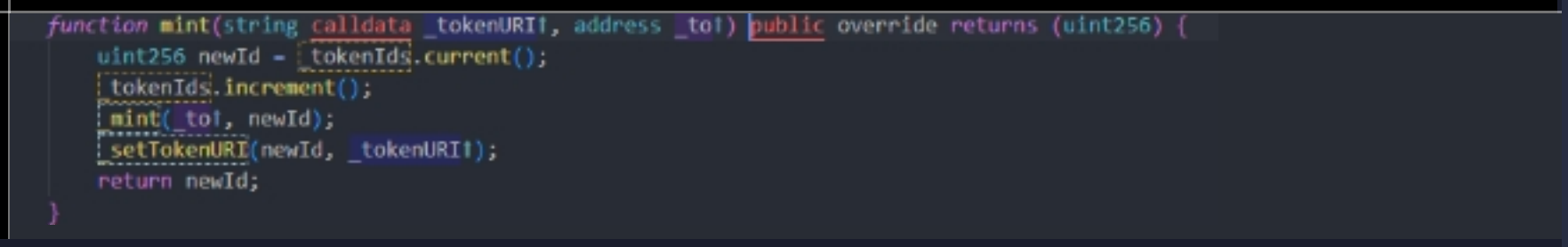
Please make sure to set appropriate access control so that only token owner/artist/owner can call the setRoyaltyArtist() which will make sure that no malicious user can call it and take advantage of royalty.

Status

Resolved

Low Severity Issues

B.2: No validity checks for tokenURI passed in mint() function

Line	Function - mint
67-73	

```
function mint(string calldata _tokenURI1, address _to1) public override returns (uint256) {
    uint256 newId = tokenIds.current();
    tokenIds.increment();
    mint(_to1, newId);
    setTokenURI(newId, _tokenURI1);
    return newId;
}
```

Description

mint() function in NFT contract takes _tokenURI but there is validity that it has been passed. So NFT can be minted without URI which can cause problems if delivered and can create problems with tokenIds.

Remediation

Please make sure to validate that the correct tokenURI has been passed.

Status

Resolved

Informational Issues

B.3: Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same.

Status

Resolved

C. Contract - Marketplace

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

C.1 Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same.

Status

Resolved

C.2: Wrong data is emitted in an event

Line	Function - mint
136, 137	`TokenInfo.owner = _buyer;` `emit Nftbought(_tokenId, TokenInfo.owner, _buyer, sellingPrice);`

Description

In Marketplace contract there is an event Nftbought which emits as
event Nftbought(uint256 indexed _tokenid, address indexed _seller, address indexed _buyer, uint256 _price);
But when it is emitted in the buy() function at that time the second parameter is seller but it emits as the buyer because data is set first.

Remediation

To mitigate the issue please set TokenInfo.owner in separate variable and put that in the event

Status

Resolved

C.3 General Recommendation

listtokenforsale() is having spelling mistakes.
It should be in its plural form: listtokensforsale()

Status

Resolved

D. Contract - GalleryFactory

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

D.1: Missing zero address check in changeNFTAddress() and changeMarketAddress()

Line	Function - mint

130-132	`function changeNftAddress(address newnft1) public override onlyOwner {`
	`  NFT = INFT(newnft1);`
	`}`

137-139	`function changeMarketAddress(address newMarket1) public override onlyOwner {`
	`  marketPlace = IMarketPlace(newMarket1);`
	`}`

Description

In GalleryFactory contract changeNftAddress() and changeMarketAddress() change NFT and Marketplace address respectively. But it is quite possible that by mistake zero addresses get set.

Remediation

Please add zero address check in changeNftAddress() and changeMarketAddress()

Status

Resolved

Informational Issues

D.2 : Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same.

Status

Resolved

Functional Testing

- Should Not Be Able To Change Royalty Fees (380Ms)
- Checking If It Is Possible To Verify Airdrop By Malicious User (539Ms)
- Should Be Minting To Gallery (70Ms)
- Checking Mintandairdrop() (253Ms)
- Checking Manageairdrop() (253Ms)
- Should Be Able To Mint (77Ms)
- Should Not Be Able To Burn NFT By Anyone (50Ms)
- Should Be Able To Burn NFT Contract
- Should Be Able To Check Nft
- Should Be Able Set Royalty
- Should Not Be Able Set Royalty By Anyone
- Should Be Able Return Token URI
- Should Be Able Return Owner Of NFT
- Should Not Allow Empty Tokenuri
- Should Be Able Return Tokeninfo
- Should Be Able Change Gallery Fee
- Should Be Able Get Latest MATIC Price (2920Ms)
- Should Be Able Get Latest Price In MATIC (408Ms)

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.

Closing Summary

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

Some issues of 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 ArtSwap NFT 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 ArtSwap NFT 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.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2022

For

ARTSWAP

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com