Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Flinch NFT Token
Website: https://flinchthemovie.com
Platform: Ethereum
Language: Solidity
Date: May 25th, 2022

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Flinch NFT team to perform the Security audit of the Flinch NFT smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 25th, 2022.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- Flinch NFTs intend to advance the art and craft of storytelling in Web3 by utilizing airdrops, staking and gamification to create valuable experiences for holders.
- The Flinch NFT is a NFT token contract which allows withdraw, mint, burn.
- The Flinch NFT contract inherits the Ownable, MerkleProof standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for Flinch NFT Token Smart Contract**
Platform	**Ethereum / Solidity**
File	Flinch.sol
File MD5 Hash	4899301EC39C1EC1753E2ABA1CD0A6BD
Updated MD5 Hash	95E163D5611B39B18D67A054D846C212
Audit Date	May 25th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Flinch NFT • Symbol: FLN • Maximum Supply: 9999 • Maximum Public Mint: 6 • Maximum Hitlist Mint: 3 • Public Sale Price: 0.065 ETH • Hitlist Sale Price: 0.05 ETH	**YES, This is valid.**
Ownership Control: • Owner can set the token URI and PlaceHolder URI. • Owner can set a merkle root value.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **“Secured”**. This token contract does contain owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 0 low and some very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in Flinch NFT Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Flinch NFT Token.

The Flinch NFT Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **not well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Flinch NFT Token smart contract code in the form of a Github weblink. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. But the code is straightforward so it is easy to understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://flinchthemovie.com which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	_startTokenId	internal	Passed	No Issue
3	totalSupply	read	Passed	No Issue
4	_totalMinted	internal	Passed	No Issue
5	supportsInterface	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	_numberMinted	internal	Passed	No Issue
8	_numberBurned	internal	Passed	No Issue
9	_getAux	internal	Passed	No Issue
10	_setAux	internal	Passed	No Issue
11	_ownershipOf	internal	Passed	No Issue
12	ownerOf	read	Passed	No Issue
13	name	read	Passed	No Issue
14	symbol	read	Passed	No Issue
15	tokenURI	read	Passed	No Issue
16	_baseURI	internal	Passed	No Issue
17	approve	write	Passed	No Issue
18	getApproved	read	Passed	No Issue
19	setApprovalForAll	write	Passed	No Issue
20	isApprovedForAll	read	Passed	No Issue
21	transferFrom	write	Passed	No Issue
22	safeTransferFrom	write	Passed	No Issue
23	safeTransferFrom	write	Passed	No Issue
24	_exists	internal	Passed	No Issue
25	_safeMint	internal	Passed	No Issue
26	_mint	internal	Passed	No Issue
27	_transfer	write	Passed	No Issue
28	_burn	internal	Passed	No Issue
29	_approve	write	Passed	No Issue
30	_checkContractOnERC721Received	write	Passed	No Issue
31	_beforeTokenTransfers	internal	Passed	No Issue
32	_afterTokenTransfers	internal	Passed	No Issue
33	owner	read	Passed	No Issue
34	onlyOwner	modifier	Passed	No Issue
35	renounceOwnership	write	access only Owner	No Issue
36	transferOwnership	write	access only Owner	No Issue
37	_transferOwnership	internal	Passed	No Issue
38	callerIsUser	modifier	Passed	No Issue
39	mint	external	Passed	No Issue
40	hitlistMint	external	Irrelevant error message	Refer Audit Findings

41	teamMint	external	access only Owner	No Issue
42	_baseURI	internal	Passed	No Issue
43	tokenURI	read	Passed	No Issue
44	setTokenUri	external	access only Owner	No Issue
45	setPlaceHolderUri	external	access only Owner	No Issue
46	setMerkleRoot	external	access only Owner	No Issue
47	getMerkleRoot	external	Passed	No Issue
48	togglePause	external	access only Owner	No Issue
49	togglehitlistSale	external	access only Owner	No Issue
50	togglePublicSale	external	access only Owner	No Issue
51	toggleReveal	external	access only Owner	No Issue
52	withdraw	external	Hard coded Values, Extra transfer code while withdraw	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Hard coded Values:

```
  function withdraw() external onlyOwner{
    //70% to utility/Flinch Franchise Project Wallet
    uint256 withdrawAmount_70 = address(this).balance * 70/100;
    //25% to inverstor/Arbor Pictures Wallet
    uint256 withdrawAmount_25 = address(this).balance * 25/100;
    //5% to project/Community Wallet
    uint256 withdrawAmount_5 = address(this).balance  * 5/100;
    payable(0x1333e81C131e1D1D0E8Bd42ecA5E45aCd0cE1De3).transfer(withdrawAmount_70);
    payable(0x08bDc77727433Bb7507D782Cb1a4aBa35987659f).transfer(withdrawAmount_25);
    payable(0x10C8C5F712101d1C285C7DF88b777565ED9C7431).transfer(withdrawAmount_5);
    payable(msg.sender).transfer(address(this).balance);
}
```

In the withdraw function there are sets of 3 wallet addresses as hard coded.

Resolution: Deployer has to confirm before deploying the contract to production.

(2) Irrelevant error message:

```
function hitlistMint(bytes32[] memory _merkleProof, uint256 _quantity) external payable callerIsUser{
    require(hitlistSale, "Flinch NFT :: Minting is on Pause");
    require((totalSupply() + _quantity) <= MAX_SUPPLY, "Flinch NFT :: Cannot mint beyond max supply");
    require((totalHitlistMint[msg.sender] + _quantity)  <= MAX_HITLIST_MINT, "Flinch NFT :: Cannot mint beyond whitelist max
    require(msg.value >= (HITLIST_SALE_PRICE * _quantity), "Flinch NFT :: Payment is below the price");
    //create leaf node
    bytes32 sender = keccak256(abi.encodePacked(msg.sender));
    require(MerkleProof.verify(_merkleProof, merkleRoot, sender), "Flinch NFT :: You're not on the Hitlist.");

    totalHitlistMint[msg.sender] += _quantity;
    _safeMint(msg.sender, _quantity);
}
```

Irrelevant error message in error message in Functions hitlistMint(). “whitelist” word should be hitelist.

Resolution: We suggest using relevant words.

(3) Unused variable:

There is a "pause" variable defined and set by the owner but not used anywhere.

Resolution: We suggest removing unused variables from the code.

(4) Extra transfer code while withdraw:

```
  function withdraw() external onlyOwner{
    //70% to utility/Flinch Franchise Project Wallet
    uint256 withdrawAmount_70 = address(this).balance * 70/100;
    //25% to inverstor/Arbor Pictures Wallet
    uint256 withdrawAmount_25 = address(this).balance * 25/100;
    //5% to project/Community Wallet
    uint256 withdrawAmount_5 = address(this).balance  * 5/100;
    payable(0x1333e81C131e1D1D0E8Bd42ecA5E45aCd0cE1De3).transfer(withdrawAmount_70);
    payable(0x08bDc77727433Bb7507D782Cb1a4aBa35987659f).transfer(withdrawAmount_25);
    payable(0x10C8C5F712101d1C285C7DF88b777565ED9C7431).transfer(withdrawAmount_5);
    payable(msg.sender).transfer(address(this).balance);
  }
```

In the withdraw function, 100% balance has been distributed among 3 wallets. So after that there will be nothing to transfer to the owner.

Resolution: We suggest removing the last line of the withdraw function.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- teamMint: Owner can mint from address.
- setTokenUri: Owner can set token URI.
- setPlaceHolderUri: Owner can set placeholder URI.
- setMerkleRoot: Owner can set merkle root value.
- togglePause: Owner can toggle pause status.
- togglehitlistSale: Owner can toggle hitlist sale status.
- togglePublicSale: Owner can toggle public sale status.
- toggleReveal: Owner can toggle reveal status.
- withdraw: Owner can withdraw the token.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We have not observed major issues. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

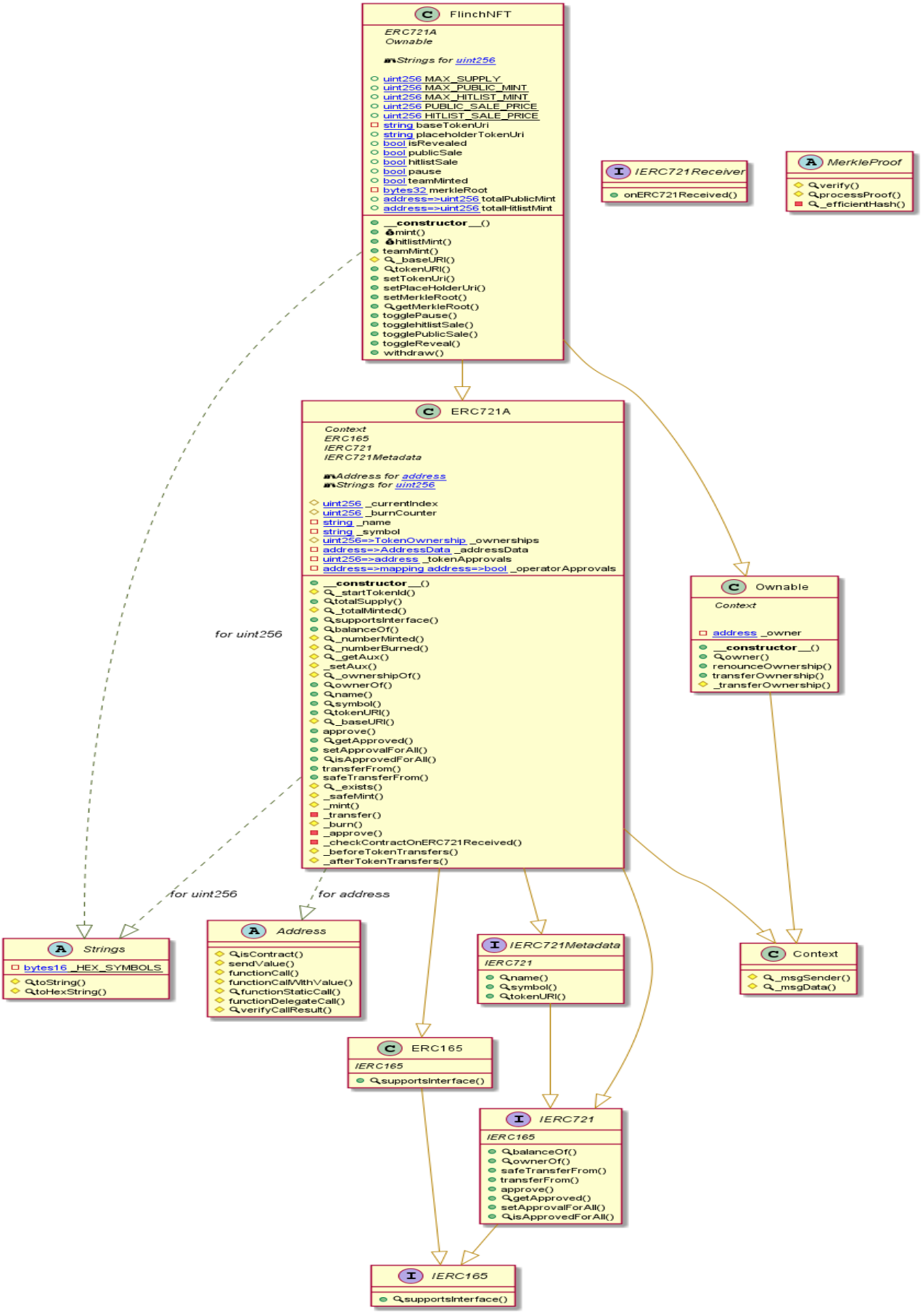
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Flinch NFT Token



Slither Results Log

Slither Log >> Flinch.sol

```
INFO:Detectors:
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).retval (Flinch.sol#565)' in ERC721A._checkCont
tOnERC721Received(address,address,uint256,bytes) (Flinch.sol#559-576) potentially used before declaration: retval == IERC721Re
ver(to).onERC721Received.selector (Flinch.sol#566)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).reason (Flinch.sol#567)' in ERC721A._checkCont
tOnERC721Received(address,address,uint256,bytes) (Flinch.sol#559-576) potentially used before declaration: reason.length == 0
inch.sol#568)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).reason (Flinch.sol#567)' in ERC721A._checkCont
tOnERC721Received(address,address,uint256,bytes) (Flinch.sol#559-576) potentially used before declaration: revert(uint256,uint
)(32 + reason,mload(uint256)(reason)) (Flinch.sol#572)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (Flinch.sol#139-157) uses assembly
        - INLINE ASM (Flinch.sol#149-152)
ERC721A._checkContractOnERC721Received(address,address,uint256,bytes) (Flinch.sol#559-576) uses assembly
        - INLINE ASM (Flinch.sol#571-573)
MerkleProof._efficientHash(bytes32,bytes32) (Flinch.sol#613-619) uses assembly
        - INLINE ASM (Flinch.sol#614-618)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (Flinch.sol#76-78) is never used and should be removed
Address.functionCall(address,bytes,string) (Flinch.sol#80-86) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Flinch.sol#88-94) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Flinch.sol#96-107) is never used and should be removed
Address.functionDelegateCall(address,bytes) (Flinch.sol#124-126) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (Flinch.sol#128-137) is never used and should be removed
Address.functionStaticCall(address,bytes) (Flinch.sol#109-111) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Flinch.sol#113-122) is never used and should be removed
Address.sendValue(address,uint256) (Flinch.sol#69-74) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (Flinch.sol#139-157) is never used and should be removed
Context._msgData() (Flinch.sol#59-61) is never used and should be removed
ERC721A._baseURI() (Flinch.sol#347-349) is never used and should be removed
ERC721A._burn(uint256) (Flinch.sol#501-503) is never used and should be removed
ERC721A._burn(uint256,bool) (Flinch.sol#505-548) is never used and should be removed
ERC721A._getAux(address) (Flinch.sol#297-299) is never used and should be removed
ERC721A._numberBurned(address) (Flinch.sol#293-295) is never used and should be removed
```

```
ERC721A._numberMinted(address) (Flinch.sol#289-291) is never used and should be removed
ERC721A._setAux(address,uint64) (Flinch.sol#301-303) is never used and should be removed
ERC721A._totalMinted() (Flinch.sol#271-275) is never used and should be removed
FlinchNFT._baseURI() (Flinch.sol#719-721) is never used and should be removed
Strings.toHexString(uint256) (Flinch.sol#29-40) is never used and should be removed
Strings.toHexString(uint256,uint256) (Flinch.sol#42-52) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (Flinch.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Flinch.sol#69-74):
        - (success) = recipient.call{value: amount}() (Flinch.sol#72)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Flinch.sol#96-107):
        - (success,returndata) = target.call{value: value}(data) (Flinch.sol#105)
Low level call in Address.functionStaticCall(address,bytes,string) (Flinch.sol#113-122):
        - (success,returndata) = target.staticcall(data) (Flinch.sol#120)
Low level call in Address.functionDelegateCall(address,bytes,string) (Flinch.sol#128-137):
        - (success,returndata) = target.delegatecall(data) (Flinch.sol#135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ERC721A.safeTransferFrom(address,address,uint256,bytes)._data (Flinch.sol#399) is not in mixedCase
Variable ERC721A._currentIndex (Flinch.sol#239) is not in mixedCase
Variable ERC721A._burnCounter (Flinch.sol#241) is not in mixedCase
Variable ERC721A._ownerships (Flinch.sol#247) is not in mixedCase
Parameter FlinchNFT.mint(uint256)._quantity (Flinch.sol#690) is not in mixedCase
Parameter FlinchNFT.hitlistMint(bytes32[],uint256)._merkleProof (Flinch.sol#700) is not in mixedCase
Parameter FlinchNFT.hitlistMint(bytes32[],uint256)._quantity (Flinch.sol#700) is not in mixedCase
Parameter FlinchNFT.setTokenUri(string)._baseTokenUri (Flinch.sol#736) is not in mixedCase
Parameter FlinchNFT.setPlaceHolderUri(string)._placeholderTokenUri (Flinch.sol#739) is not in mixedCase
Parameter FlinchNFT.setMerkleRoot(bytes32)._merkleRoot (Flinch.sol#743) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
balanceOf(address) should be declared external:
        - ERC721A.balanceOf(address) (Flinch.sol#284-287)
name() should be declared external:
        - ERC721A.name() (Flinch.sol#332-334)
symbol() should be declared external:
        - ERC721A.symbol() (Flinch.sol#336-338)
tokenURI(uint256) should be declared external:
        - ERC721A.tokenURI(uint256) (Flinch.sol#340-345)
        - FlinchNFT.tokenURI(uint256) (Flinch.sol#724-734)
approve(address,uint256) should be declared external:
        - ERC721A.approve(address,uint256) (Flinch.sol#351-360)
setApprovalForAll(address,bool) should be declared external:
        - ERC721A.setApprovalForAll(address,bool) (Flinch.sol#368-373)
transferFrom(address,address,uint256) should be declared external:
        - ERC721A.transferFrom(address,address,uint256) (Flinch.sol#379-385)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721A.safeTransferFrom(address,address,uint256) (Flinch.sol#387-393)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Flinch.sol#639-641)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Flinch.sol#643-646)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Flinch.sol analyzed (12 contracts with 75 detectors), 57 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Flinch.sol

Security

Transaction origin:

Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.

more

Pos: 686:16:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 529:45:

Gas & Economy

Gas costs:

Gas requirement of function ERC721A.ownerOf is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 328:4:

Gas costs:

Gas requirement of function FlinchNFT.withdraw is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 766:6:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 602:8:

Miscellaneous

Email: audit@EtherAuthority.io

Constant/View/Pure functions:

MerkleProof._efficientHash(bytes32,bytes32) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 613:4:

Similar variable names:

FlinchNFT.withdraw() : Variables have very similar names "withdrawAmount_70" and "withdrawAmount_25". Note: Modifiers are currently not considered by this static analysis.

Pos: 774:69:

Similar variable names:

FlinchNFT.withdraw() : Variables have very similar names "withdrawAmount_70" and "withdrawAmount_5". Note: Modifiers are currently not considered by this static analysis.

Pos: 768:8:

No return:

MerkleProof._efficientHash(bytes32,bytes32): Defines a return type but never explicitly returns a value.

Pos: 613:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 701:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 725:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 768:36:

Solhint Linter

Flinch.sol

```
Flinch.sol:266:18: Error: Parse error: missing ';' at '{'
Flinch.sol:272:18: Error: Parse error: missing ';' at '{'
Flinch.sol:308:18: Error: Parse error: missing ';' at '{'
Flinch.sol:435:18: Error: Parse error: missing ';' at '{'
Flinch.sol:479:18: Error: Parse error: missing ';' at '{'
Flinch.sol:522:18: Error: Parse error: missing ';' at '{'
Flinch.sol:542:48: Error: Parse error: mismatched input ';' expecting
'('
Flinch.sol:545:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io