Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: The Holy Vessel-Pot Pass
Website: www.plutuslandglobal.com
Platform: Ethereum
Language: Solidity
Date: September 13th, 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 The Holy Vessel-Pot Pass Token team to perform the Security audit of the The Holy Vessel-Pot Pass Token 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 September 13th, 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

- The Holy Vessel-Pot Pass is an ERC721 contract in which any user can mint a NFT only once. Special address wallet, set by the owner, can mint 100 NFT.
- The Holy Vessel-Pot Pass contract inherits the ERC721, ERC721Enumerable, Address, Counters, Pausable, 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 The Holy Vessel-Pot Pass Token Smart Contract**
Platform	**Ethereum / Solidity**
File	PremintCreatorKey.sol
File MD5 Hash	5282FFD4622C55F99825DF0CE4927642
Audit Date	September 13th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: The Holy Vessel-Pot Pass ● Symbol: The Holy Vessel-Pot Pass ● Unlimited token minting possibility	**YES, This is valid.**
Other Specifications ● Open Zeppelin standard code is used. ● Owner can get a baseURI and a special address. ● Owner can set a baseURI and a special address.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Secured"**. Also, these contracts do contain owner control, which does not make them 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	Moderated
	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	Moderated
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 the The Holy Vessel-Pot Pass 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 The Holy Vessel-Pot Pass Token.

The Holy Vessel-Pot PassToken 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 used, which is a good thing.

Documentation

We were given The Holy Vessel-Pot Pass Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented but the logic is straightforward. So it is easy to quickly 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://www.plutuslandglobal.com/ which provided rich information about the project architecture.

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 not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Unused internal variable	Refer Audit Findings
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	whenNotPaused	modifier	Passed	No Issue
7	whenPaused	modifier	Passed	No Issue
8	paused	read	Passed	No Issue
9	_requireNotPaused	internal	Passed	No Issue
10	_requirePaused	internal	Passed	No Issue
11	_pause	internal	Passed	No Issue
12	_unpause	internal	Passed	No Issue
13	supportsInterface	read	Passed	No Issue
14	balanceOf	read	Passed	No Issue
15	ownerOf	read	Passed	No Issue
16	name	read	Passed	No Issue
17	symbol	read	Passed	No Issue
18	tokenURI	read	Passed	No Issue
19	_baseURI	internal	Passed	No Issue
20	approve	write	Passed	No Issue
21	getApproved	read	Passed	No Issue
22	setApprovalForAll	write	Passed	No Issue
23	isApprovedForAll	read	Passed	No Issue
24	transferFrom	write	Passed	No Issue
25	safeTransferFrom	write	Passed	No Issue
26	safeTransferFrom	write	Passed	No Issue
27	_ownerOf	internal	Passed	No Issue
28	_safeTransfer	internal	Passed	No Issue
29	_exists	internal	Passed	No Issue
30	_isApprovedOrOwner	internal	Passed	No Issue
31	_safeMint	internal	Passed	No Issue
32	_safeMint	internal	Passed	No Issue
33	_mint	internal	Passed	No Issue
34	_burn	internal	Passed	No Issue
35	_transfer	internal	Passed	No Issue
36	_approve	internal	Passed	No Issue
37	_setApprovalForAll	internal	Passed	No Issue
38	_requireMinted	internal	Passed	No Issue
39	_checkOnERC721Received	write	Passed	No Issue
40	_beforeTokenTransfer	internal	Passed	No Issue
41	_afterTokenTransfer	internal	Passed	No Issue

42	_beforeConsecutiveTokenTransfer	internal	Passed	No Issue
43	_afterConsecutiveTokenTransfer	internal	Passed	No Issue
44	supportsInterface	read	Passed	No Issue
45	tokenOfOwnerByIndex	read	Passed	No Issue
46	totalSupply	read	Passed	No Issue
47	tokenByIndex	read	Passed	No Issue
48	_beforeTokenTransfer	internal	Passed	No Issue
49	_addTokenToOwnerEnumeration	write	Passed	No Issue
50	_addTokenToAllTokensEnumeration	write	Passed	No Issue
51	_removeTokenFromOwnerEnumeration	write	Passed	No Issue
52	_removeTokenFromAllTokensEnumeration	write	Passed	No Issue
53	mint	external	Set appropriate error message	Refer Audit Findings
54	_multimint	write	Passed	No Issue
55	pause	external	access only Owner	No Issue
56	unpause	external	access only Owner	No Issue
57	totalSupply	read	Wrong value return	Refer Audit Findings
58	tokenURI	read	Passed	No Issue
59	getBaseURI	read	Passed	No Issue
60	setBaseURI	write	Critical operation lacks event log	Refer Audit Findings
61	_baseURI	internal	Passed	No Issue
62	supportsInterface	read	Passed	No Issue
63	_beforeTokenTransfer	internal	Passed	No Issue
64	setSpecialAddress	write	Critical operation lacks event log	Refer Audit Findings
65	getSpecialAddress	read	access only Owner	No Issue
66	getnftamount	read	Passed	No Issue
67	nftAdd	internal	Passed	No Issue
68	nftSub	internal	Unused internal function	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) Set appropriate error message:

```
function mint(bytes32[] calldata _merkleProof) external whenNotPaused {

  address to=msg.sender;
  require(
      !userStatus[to]
    ,
    "repeat mint"
  );
  bytes32 leaf = keccak256(abi.encodePacked(to));
    require(
      MerkleProof.verify(_merkleProof, merkle ,leaf)
    ,
    "aaa"
  );
```

The “required” with an irrelevant error message in mint() function.

Resolution: We suggest writing appropriate error messages to get the failure of the transaction.

(2) Critical operation lacks event log:

Missing event log for:

- setBaseURI()
- setSpecialAddress()

Resolution: Please write an event log for listed events.

(3) Unused internal function / variable:

```
function nftSub(uint256 a) internal pure returns (uint256) {
    a--;
    return a;
}
```

nftSub function has been defined as internal but not used anywhere.

```
address private ownera;

bytes32 private merkle=0xee963cd8daef5f5c1fd8d06e8b9780667c51da43a0719721b1edfb45

constructor() ERC721("The Holy Vessel-Pot Pass", "The Holy Vessel-Pot Pass") {
  baseURI = "https://ipfs.io/ipfs/bafybeibroqistiqyxg2p3sdnjintj6qgwle4acdl6ws5mw
  nextTokenId.increment();
  ownera=msg.sender;
  //beneficiary = payable(msg.sender);
}
```

ownera variable has been defined as private and set in constructor but never used.

Resolution: We suggest removing unused variables and functions.

(4) Wrong value return:

```
// @notice Returns the total number of mints
function totalSupply() public view override returns (uint256) {
  return nextTokenId.current();
}
```

Comment shows that totalSupply should return the total number of tokens minted, but actually it returns the next token number to be minted.

Resolution: We suggest either changing the comment or correct the return value by totalSupply function.

(5) Initialized by default value:

```
string private baseURI = "";

Counters.Counter private nextTokenId;
mapping(address => bool) private userStatus;
address private specialAddress=0x522c6F7BC1f43326AABFefa72bA5D8DEe9C37F08;
uint256 private nowTotalCount=0;
address private ownera;
```

These 2 variables have been initialized by default value.

Resolution: We suggest not to initialize by default value. It will save some gas.

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:

- pause: Owner can trigger stopped state.
- unpause: Owner can return to normal state.
- setBaseURI: Owner can allow to change the baseURI.
- setSpecialAddress: Owner can set a special address.
- getSpecialAddress: Owner can get a special address.

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 in the form of a file and we have used all possible tests based on given objects as files. We have not observed any major issues in the token smart contract. So, **it's good to go for the mainnet deployment**.

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.

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

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

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).

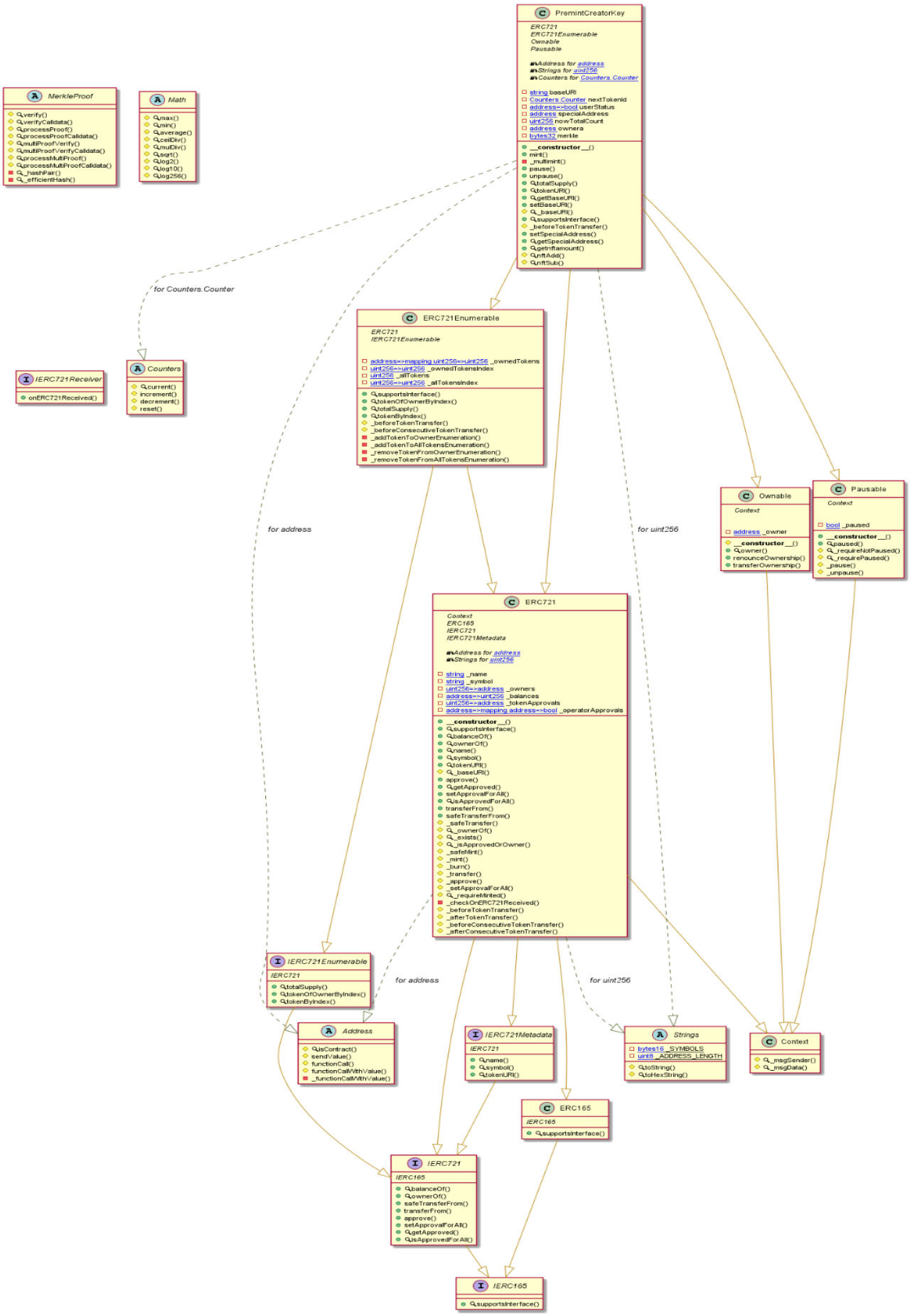
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - The Holy Vessel-Pot Pass Token



Slither Results Log

Slither Log >> PremintCreatorKey.sol

```
INFO:Detectors:
PremintCreatorKey.setSpecialAddress(address)._specialAddress (PremintCreatorKey.sol#1947) lacks a zero-check on :
                - specialAddress = _specialAddress (PremintCreatorKey.sol#1948)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (PremintCreatorKey.sol#1554)' in ERC721._checkOn
ERC721Received(address,address,uint256,bytes) (PremintCreatorKey.sol#1547-1569) potentially used before declaration: retval ==
 IERC721Receiver.onERC721Received.selector (PremintCreatorKey.sol#1555)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (PremintCreatorKey.sol#1556)' in ERC721._checkOn
ERC721Received(address,address,uint256,bytes) (PremintCreatorKey.sol#1547-1569) potentially used before declaration: reason.le
ngth == 0 (PremintCreatorKey.sol#1557)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (PremintCreatorKey.sol#1556)' in ERC721._checkOn
ERC721Received(address,address,uint256,bytes) (PremintCreatorKey.sol#1547-1569) potentially used before declaration: revert(ui
nt256,uint256)(32 + reason,mload(uint256)(reason)) (PremintCreatorKey.sol#1562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Reentrancy in PremintCreatorKey._multimint(address,uint256) (PremintCreatorKey.sol#1857-1870):
        External calls:
        - _safeMint(to,startTokenId) (PremintCreatorKey.sol#1865)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (PremintCreatorKey.sol#1554-1565)
        State variables written after the call(s):
        - nowTotalCount = nftAdd(nowTotalCount) (PremintCreatorKey.sol#1867)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
MerkleProof._efficientHash(bytes32,bytes32) (PremintCreatorKey.sol#188-195) uses assembly
        - INLINE ASM (PremintCreatorKey.sol#190-194)
Math.mulDiv(uint256,uint256,uint256) (PremintCreatorKey.sol#273-353) uses assembly
        - INLINE ASM (PremintCreatorKey.sol#284-288)
        - INLINE ASM (PremintCreatorKey.sol#304-311)
        - INLINE ASM (PremintCreatorKey.sol#318-327)
Strings.toString(uint256) (PremintCreatorKey.sol#571-591) uses assembly
        - INLINE ASM (PremintCreatorKey.sol#577-579)
        - INLINE ASM (PremintCreatorKey.sol#583-585)
Address.isContract(address) (PremintCreatorKey.sol#643-654) uses assembly
        - INLINE ASM (PremintCreatorKey.sol#650-652)
Address._functionCallWithValue(address,bytes,uint256,string) (PremintCreatorKey.sol#751-777) uses assembly
        - INLINE ASM (PremintCreatorKey.sol#769-772)
```

```
ERC721._checkOnERC721Received(address,address,uint256,bytes) (PremintCreatorKey.sol#1547-1569) uses assembly
        - INLINE ASM (PremintCreatorKey.sol#1561-1563)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (PremintCreatorKey.sol#751-777) is never used and should be remov
ed
Address.functionCall(address,bytes) (PremintCreatorKey.sol#698-700) is never used and should be removed
Address.functionCall(address,bytes,string) (PremintCreatorKey.sol#708-714) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (PremintCreatorKey.sol#727-733) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (PremintCreatorKey.sol#741-749) is never used and should be remove
d
Address.sendValue(address,uint256) (PremintCreatorKey.sol#672-678) is never used and should be removed
Context._msgData() (PremintCreatorKey.sol#983-986) is never used and should be removed
Counters.decrement(Counters.Counter) (PremintCreatorKey.sol#215-221) is never used and should be removed
Counters.reset(Counters.Counter) (PremintCreatorKey.sol#223-225) is never used and should be removed
ERC721._afterConsecutiveTokenTransfer(address,address,uint256,uint96) (PremintCreatorKey.sol#1633-1638) is never used and shou
ld be removed
ERC721._baseURI() (PremintCreatorKey.sol#1223-1225) is never used and should be removed
ERC721._beforeConsecutiveTokenTransfer(address,address,uint256,uint96) (PremintCreatorKey.sol#1615-1627) is never used and sho
uld be removed
ERC721._burn(uint256) (PremintCreatorKey.sol#1439-1460) is never used and should be removed
Math.average(uint256,uint256) (PremintCreatorKey.sol#252-255) is never used and should be removed
Math.ceilDiv(uint256,uint256) (PremintCreatorKey.sol#263-266) is never used and should be removed
Math.log10(uint256) (PremintCreatorKey.sol#476-508) is never used and should be removed
Math.log10(uint256,Math.Rounding) (PremintCreatorKey.sol#514-519) is never used and should be removed
Math.log2(uint256) (PremintCreatorKey.sol#423-459) is never used and should be removed
Math.log2(uint256,Math.Rounding) (PremintCreatorKey.sol#465-470) is never used and should be removed
Math.log256(uint256) (PremintCreatorKey.sol#527-551) is never used and should be removed
Math.log256(uint256,Math.Rounding) (PremintCreatorKey.sol#557-562) is never used and should be removed
Math.max(uint256,uint256) (PremintCreatorKey.sol#237-239) is never used and should be removed
Math.min(uint256,uint256) (PremintCreatorKey.sol#244-246) is never used and should be removed
Math.mulDiv(uint256,uint256,uint256) (PremintCreatorKey.sol#273-353) is never used and should be removed
Math.mulDiv(uint256,uint256,uint256,Math.Rounding) (PremintCreatorKey.sol#358-369) is never used and should be removed
Math.sqrt(uint256) (PremintCreatorKey.sol#376-407) is never used and should be removed
Math.sqrt(uint256,Math.Rounding) (PremintCreatorKey.sol#412-417) is never used and should be removed
MerkleProof.multiProofVerify(bytes32[],bool[],bytes32,bytes32[]) (PremintCreatorKey.sol#67-74) is never used and should be rem
oved
```

```
MerkleProof.multiProofVerifyCalldata(bytes32[],bool[],bytes32,bytes32[]) (PremintCreatorKey.sol#81-88) is never used and shoul
d be removed
MerkleProof.processMultiProof(bytes32[],bool[],bytes32[]) (PremintCreatorKey.sol#97-136) is never used and should be removed
MerkleProof.processMultiProofCalldata(bytes32[],bool[],bytes32[]) (PremintCreatorKey.sol#143-182) is never used and should be
removed
MerkleProof.processProofCalldata(bytes32[],bytes32) (PremintCreatorKey.sol#53-59) is never used and should be removed
MerkleProof.verifyCalldata(bytes32[],bytes32,bytes32) (PremintCreatorKey.sol#24-30) is never used and should be removed
PremintCreatorKey.nftSub(uint256) (PremintCreatorKey.sol#1971-1974) is never used and should be removed
Strings.toHexString(address) (PremintCreatorKey.sol#620-622) is never used and should be removed
Strings.toHexString(uint256) (PremintCreatorKey.sol#596-600) is never used and should be removed
Strings.toHexString(uint256,uint256) (PremintCreatorKey.sol#605-615) is never used and should be removed
Strings.toString(uint256) (PremintCreatorKey.sol#571-591) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version>=0.8.4 (PremintCreatorKey.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.1
2/0.7.6
solc-0.8.4 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) (PremintCreatorKey.sol#672-678):
        - (success) = recipient.call{value: amount}() (PremintCreatorKey.sol#676)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (PremintCreatorKey.sol#751-777):
        - (success,returndata) = target.call{value: weiValue}(data) (PremintCreatorKey.sol#760)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter PremintCreatorKey.mint(bytes32[])._merkleProof (PremintCreatorKey.sol#1831) is not in mixedCase
Parameter PremintCreatorKey.setBaseURI(string)._uri (PremintCreatorKey.sol#1913) is not in mixedCase
Parameter PremintCreatorKey.setSpecialAddress(address)._specialAddress (PremintCreatorKey.sol#1947) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (PremintCreatorKey.sol#984)" inContext (PremintCreatorKey.sol#978-987)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
PremintCreatorKey.merkle (PremintCreatorKey.sol#1820) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (PremintCreatorKey.sol#1033-1036)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (PremintCreatorKey.sol#1042-1046)
name() should be declared external:
        - ERC721.name() (PremintCreatorKey.sol#1197-1199)
symbol() should be declared external:
        - ERC721.symbol() (PremintCreatorKey.sol#1204-1206)
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (PremintCreatorKey.sol#1211-1216)
        - PremintCreatorKey.tokenURI(uint256) (PremintCreatorKey.sol#1887-1904)
```

```
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (PremintCreatorKey.sol#1211-1216)
        - PremintCreatorKey.tokenURI(uint256) (PremintCreatorKey.sol#1887-1904)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (PremintCreatorKey.sol#1230-1240)
setApprovalForAll(address,bool) should be declared external:
        - ERC721.setApprovalForAll(address,bool) (PremintCreatorKey.sol#1254-1256)
transferFrom(address,address,uint256) should be declared external:
        - ERC721.transferFrom(address,address,uint256) (PremintCreatorKey.sol#1268-1277)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721.safeTransferFrom(address,address,uint256) (PremintCreatorKey.sol#1282-1288)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (PremintCreatorKey.sol#1664-1667)
tokenByIndex(uint256) should be declared external:
        - ERC721Enumerable.tokenByIndex(uint256) (PremintCreatorKey.sol#1679-1682)
getBaseURI() should be declared external:
        - PremintCreatorKey.getBaseURI() (PremintCreatorKey.sol#1907-1909)
setBaseURI(string) should be declared external:
        - PremintCreatorKey.setBaseURI(string) (PremintCreatorKey.sol#1913-1915)
setSpecialAddress(address) should be declared external:
        - PremintCreatorKey.setSpecialAddress(address) (PremintCreatorKey.sol#1947-1949)
getSpecialAddress() should be declared external:
        - PremintCreatorKey.getSpecialAddress() (PremintCreatorKey.sol#1951-1953)
getnftamount() should be declared external:
        - PremintCreatorKey.getnftamount() (PremintCreatorKey.sol#1955-1962)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:PremintCreatorKey.sol analyzed (17 contracts with 75 detectors), 84 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

PremintCreatorKey.sol

Gas & Economy

Gas costs:

Gas requirement of function PremintCreatorKey.mint 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: 37:2:

Gas costs:

Gas requirement of function PremintCreatorKey.pause 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: 80:2:

Gas costs:

Gas requirement of function PremintCreatorKey.getnftamount 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: 161:2:

Miscellaneous

Constant/View/Pure functions:

PremintCreatorKey._beforeTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 144:2:

Email: audit@EtherAuthority.io

Similar variable names:

PremintCreatorKey.() : Variables have very similar names "ownera" and "_owners". Note: Modifiers are currently not considered by this static analysis.
Pos: 32:4:

Similar variable names:

PremintCreatorKey.() : Variables have very similar names "ownera" and "_owner". Note: Modifiers are currently not considered by this static analysis.
Pos: 32:4:

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: 40:4:

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: 46:6:

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: 100:4:

Solhint Linter

PremintCreatorKey.sol

```
PremintCreatorKey.sol:2:1: Error: Compiler version >=0.8.9 does not
satisfy the r semver requirement
PremintCreatorKey.sol:29:3: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
```

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