Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer: Space Doge Team
Website: thespacedoge.com
Platform: Ethereum
Language: Solidity
Date: August 6th, 2021

Table of contents

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 PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by the SpaceDoge team to perform the Security audit of the SpaceDoge NFT 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 August 6th, 2021.

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 SPACE DOGE Collection is 10,000 NFTs from 24 countries carefully chosen by elite scientists led by The Space Doge Pack.

Audit scope

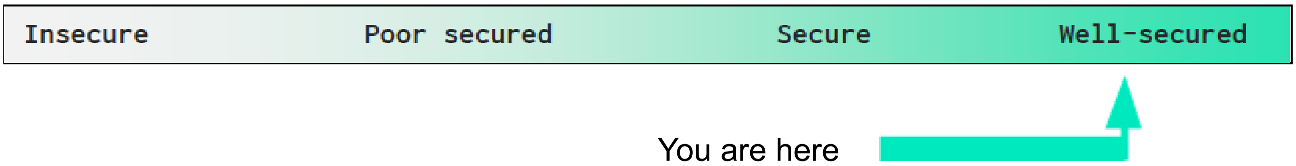
Name	Code Review and Security Analysis Report for SpaceDoge Token Smart Contract
Platform	**Ethereum / Solidity**
File	SpaceDoge.sol
Smart Contract Online Code	https://github.com/thespacedoge/protocol/blob/master/SpaceDoge.sol
File MD5 Hash	E66D6A049AC960EC441F1CDA9140E8E1
Audit Date	August 6th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: SpaceDoge	**YES, This is valid.**
Symbol: SPACEDOGE	**YES, This is valid.**
Maximum Doge Supply: 10000000000000000	**YES, This is valid.**
One Time Max Minting: 30	**YES, This is valid.**
Minting Price: 0.074 ETH	**YES, This is valid.**
• The SpaceDoge Owner can access functions like setMintPrice, setMaxToMint, reserveDoges, setRevealTimestamp, setProvenanceHash, setBaseURI, setSaleState, emergencySetStartingIndexBlock, etc.	**YES, This is valid. The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Well-secured"**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.



We used various tools like MythX, Slither 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.
These issues are fixed/acknowledged in the revised smart contract code.

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	Moderated
	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	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	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

Code Quality

This audit scope has 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in SpaceDoge 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 SpaceDoge Token token.

The SpaceDoge 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 **well** commented on smart contracts.

Documentation

We were given a SpaceDoge smart contracts code in the form of a github code. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are well commented. 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://thespacedoge.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. And their core code blocks are written well.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

SpaceDoge.sol

(1) Interface

(a) IERC165

(b) IERC721

(c) IERC721Metadata

(d) IERC721Enumerable

(e) IERC721Receiver

(2) Inherited contracts

(a) ERC721

(b) Ownable

(3) Usages

(a) using SafeMath for uint256;

(4) Functions

Sl.	Functions	Type	Observation	Conclusion
1	tokensOfOwner	external	Passed	No Issue
2	exists	read	Passed	No Issue
3	setMintPrice	external	access only Owner	No Issue
4	setMaxToMint	external	access only Owner	No Issue
5	reserveDoges	external	access only Owner	No Issue
6	setRevealTimestamp	external	access only Owner	No Issue
7	setProvenanceHash	external	access only Owner	No Issue
8	setBaseURI	external	access only Owner	No Issue
9	setSaleState	external	access only Owner	No Issue
10	mintDoges	external	Passed	No Issue
11	setStartingIndex	external	Passed	No Issue
12	emergencySetStartingIndex Block	external	access only Owner	No Issue
13	withdraw	external	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	read	Passed	No Issue

20	tokenOfOwnerByIndex	read	Passed	No Issue
21	totalSupply	read	Passed	No Issue
22	tokenByIndex	read	Passed	No Issue
23	approve	write	Passed	No Issue
24	getApproved	read	Passed	No Issue
25	setApprovalForAll	write	Passed	No Issue
26	isApprovedForAll	read	Passed	No Issue
27	transferFrom	write	Passed	No Issue
28	safeTransferFrom	write	Passed	No Issue
29	safeTransferFrom	write	Passed	No Issue
30	_safeTransfer	internal	Passed	No Issue
31	_exists	internal	Passed	No Issue
32	_isApprovedOrOwner	internal	Passed	No Issue
33	_safeMint	internal	Passed	No Issue
34	_mint	internal	Passed	No Issue
35	_burn	internal	Passed	No Issue
36	_transfer	internal	Passed	No Issue
37	_setTokenURI	internal	Passed	No Issue
38	_setBaseURI	internal	Passed	No Issue
39	_checkOnERC721Received	write	Passed	No Issue
40	_approve	internal	Passed	No Issue
41	_beforeTokenTransfer	internal	Passed	No Issue
42	owner	read	Passed	No Issue
43	onlyOwner	modifier	Passed	No Issue
44	renounceOwnership	write	access only Owner	No Issue
45	transferOwnership	write	access only Owner	No Issue

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.

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Discussion / Best practices:

(1) Use latest solidity version

```
pragma solidity ^0.7.6;
```

Although the current version does not raise any security vulnerability, it is recommended to use the latest solidity version, which is 0.8.6 at the time of this audit.

Centralization

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

- setMintPrice: The Owner can set mint price.
- setMaxToMint: The Owner can set a max price.
- reserveDoges: The Owner can check Invalid address to reserve then set totalSupply.
- setRevealTimestamp: The Owner can set reveal timestamp when finished the sale.
- setProvenanceHash: The Owner can set provenance once it's calculated.
- setBaseURI: The Owner can set baseURI.
- setSaleState: The Owner can check Pause sale if active, make active if paused.
- emergencySetStartingIndexBlock: The Owner can set the starting index block for the collection, essentially unblocking the setting starting index.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those issues are fixed in revised code. So, **it's good to go to 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 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 contract, based on standard audit procedure scope, is **"Well-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).

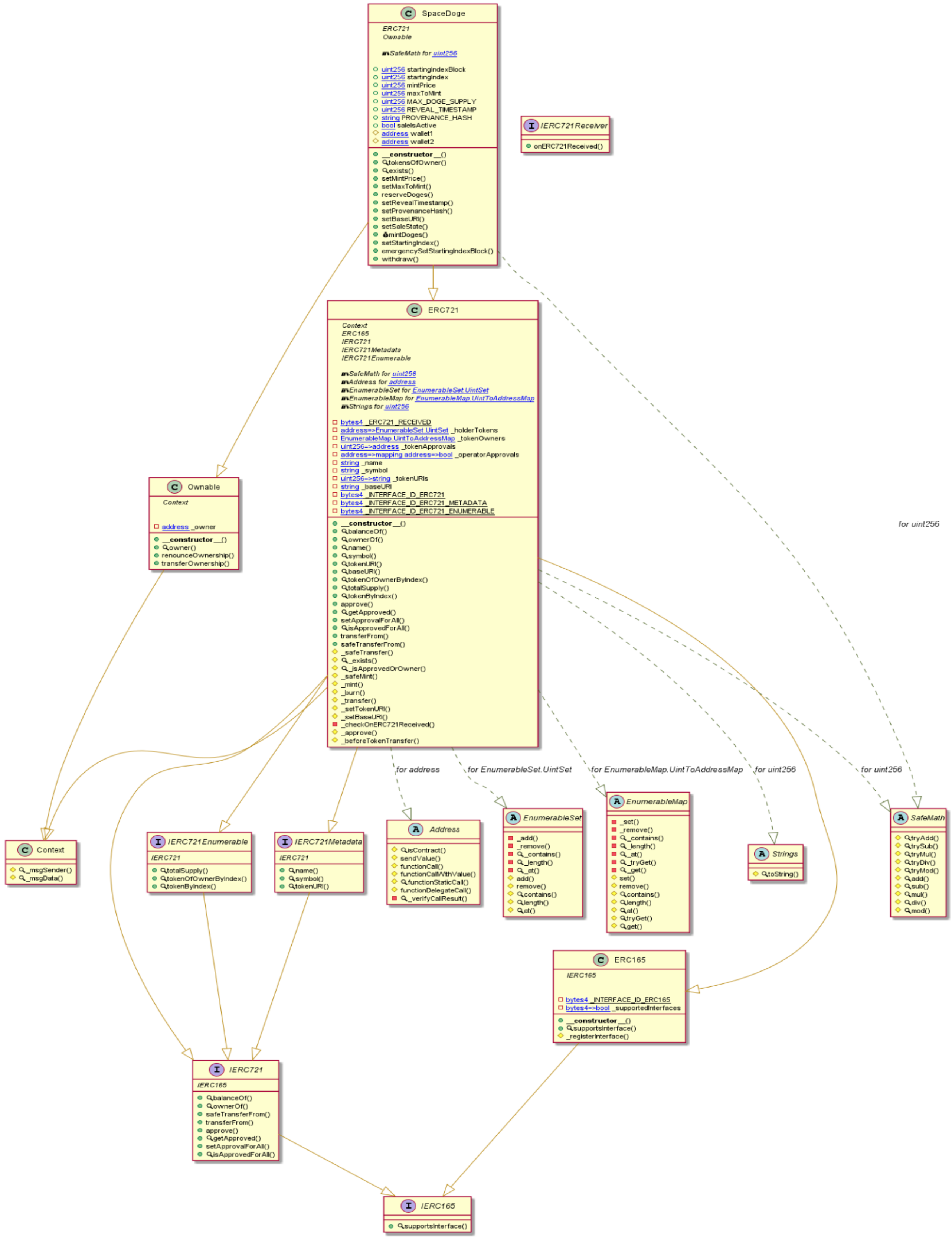
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Email: audit@EtherAuthority.io

Slither Results Log

Slither log >> SpaceDoge.sol

```
root@server:/chetan/gaza/mycontracts# slither SpaceDoge.sol
INFO:Detectors:
SpaceDoge.setStartingIndex() (SpaceDoge.sol#1946-1959) uses a weak PRNG: "startingIndex = uint256(blockhash(uint256)(block.number - 1)) %
 MAX_DOGE_SUPPLY (SpaceDoge.sol#1953)"
SpaceDoge.setStartingIndex() (SpaceDoge.sol#1946-1959) uses a weak PRNG: "startingIndex = uint256(blockhash(uint256)(startingIndexBlock))
 % MAX_DOGE_SUPPLY (SpaceDoge.sol#1950)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
SpaceDoge.setStartingIndex() (SpaceDoge.sol#1946-1959) uses a dangerous strict equality:
        - require(bool,string)(startingIndex == 0,Starting index is already set) (SpaceDoge.sol#1947)
SpaceDoge.setStartingIndex() (SpaceDoge.sol#1946-1959) uses a dangerous strict equality:
        - startingIndex == 0 (SpaceDoge.sol#1956)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
SpaceDoge.tokensOfOwner(address).index (SpaceDoge.sol#1855) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ERC721._mint(address,uint256) (SpaceDoge.sol#1603-1614) ignores return value by _holderTokens[to].add(tokenId) (SpaceDoge.sol#1609)
ERC721._mint(address,uint256) (SpaceDoge.sol#1603-1614) ignores return value by _tokenOwners.set(tokenId,to) (SpaceDoge.sol#1611)
ERC721._burn(uint256) (SpaceDoge.sol#1626-1644) ignores return value by _holderTokens[owner].remove(tokenId) (SpaceDoge.sol#1639)
ERC721._burn(uint256) (SpaceDoge.sol#1626-1644) ignores return value by _tokenOwners.remove(tokenId) (SpaceDoge.sol#1641)
ERC721._transfer(address,address,uint256) (SpaceDoge.sol#1657-1672) ignores return value by _holderTokens[from].remove(tokenId) (SpaceDog
e.sol#1666)
ERC721._transfer(address,address,uint256) (SpaceDoge.sol#1657-1672) ignores return value by _holderTokens[to].add(tokenId) (SpaceDoge.sol
#1667)
ERC721._transfer(address,address,uint256) (SpaceDoge.sol#1657-1672) ignores return value by _tokenOwners.set(tokenId,to) (SpaceDoge.sol#1
669)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
SpaceDoge.constructor(string,string,uint256,uint256,uint256).name (SpaceDoge.sol#1836) shadows:
        - ERC721.name() (SpaceDoge.sol#1391-1393) (function)
        - IERC721Metadata.name() (SpaceDoge.sol#183) (function)
SpaceDoge.constructor(string,string,uint256,uint256,uint256).symbol (SpaceDoge.sol#1836) shadows:
        - ERC721.symbol() (SpaceDoge.sol#1398-1400) (function)
        - IERC721Metadata.symbol() (SpaceDoge.sol#188) (function)
SpaceDoge.tokensOfOwner(address)._owner (SpaceDoge.sol#1849) shadows:
        - Ownable._owner (SpaceDoge.sol#1764) (state variable)
```

```
SpaceDoge.constructor(string,string,uint256,uint256,uint256).symbol (SpaceDoge.sol#1836) shadows:
        - ERC721.symbol() (SpaceDoge.sol#1398-1400) (function)
        - IERC721Metadata.symbol() (SpaceDoge.sol#188) (function)
SpaceDoge.tokensOfOwner(address)._owner (SpaceDoge.sol#1849) shadows:
        - Ownable._owner (SpaceDoge.sol#1764) (state variable)
SpaceDoge.setBaseURI(string).baseURI (SpaceDoge.sol#1910) shadows:
        - ERC721.baseURI() (SpaceDoge.sol#1428-1430) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
SpaceDoge.mintDoges(uint256) (SpaceDoge.sol#1924-1941) uses timestamp for comparisons
        Dangerous comparisons:
        - startingIndexBlock == 0 && (totalSupply() == MAX_DOGE_SUPPLY || block.timestamp >= REVEAL_TIMESTAMP) (SpaceDoge.sol#1938)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (SpaceDoge.sol#527-536) uses assembly
        - INLINE ASM (SpaceDoge.sol#534)
Address._verifyCallResult(bool,bytes,string) (SpaceDoge.sol#672-689) uses assembly
        - INLINE ASM (SpaceDoge.sol#681-684)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (SpaceDoge.sol#580-582) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (SpaceDoge.sol#605-607) is never used and should be removed
Address.functionDelegateCall(address,bytes) (SpaceDoge.sol#654-656) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (SpaceDoge.sol#664-670) is never used and should be removed
Address.functionStaticCall(address,bytes) (SpaceDoge.sol#630-632) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (SpaceDoge.sol#640-646) is never used and should be removed
Address.sendValue(address,uint256) (SpaceDoge.sol#554-560) is never used and should be removed
Context._msgData() (SpaceDoge.sol#20-23) is never used and should be removed
ERC721._burn(uint256) (SpaceDoge.sol#1626-1644) is never used and should be removed
ERC721._setTokenURI(uint256,string) (SpaceDoge.sol#1681-1684) is never used and should be removed
EnumerableMap._get(EnumerableMap.Map,bytes32) (SpaceDoge.sol#1149-1153) is never used and should be removed
EnumerableMap._remove(EnumerableMap.Map,bytes32) (SpaceDoge.sol#1067-1099) is never used and should be removed
EnumerableMap._tryGet(EnumerableMap.Map,bytes32) (SpaceDoge.sol#1136-1140) is never used and should be removed
EnumerableMap.get(EnumerableMap.UintToAddressMap,uint256) (SpaceDoge.sol#1239-1241) is never used and should be removed
EnumerableMap.remove(EnumerableMap.UintToAddressMap,uint256) (SpaceDoge.sol#1189-1191) is never used and should be removed
EnumerableMap.tryGet(EnumerableMap.UintToAddressMap,uint256) (SpaceDoge.sol#1227-1230) is never used and should be removed
EnumerableSet.add(EnumerableSet.AddressSet,address) (SpaceDoge.sol#891-893) is never used and should be removed
EnumerableSet.add(EnumerableSet.Bytes32Set,bytes32) (SpaceDoge.sol#837-839) is never used and should be removed
```

```
EnumerableMap.remove(EnumerableMap.UintToAddressMap,uint256) (SpaceDoge.sol#1189-1191) is never used and should be removed
EnumerableMap.tryGet(EnumerableMap.UintToAddressMap,uint256) (SpaceDoge.sol#1227-1230) is never used and should be removed
EnumerableSet.add(EnumerableSet.AddressSet,address) (SpaceDoge.sol#891-893) is never used and should be removed
EnumerableSet.add(EnumerableSet.Bytes32Set,bytes32) (SpaceDoge.sol#837-839) is never used and should be removed
EnumerableSet.at(EnumerableSet.AddressSet,uint256) (SpaceDoge.sol#929-931) is never used and should be removed
EnumerableSet.at(EnumerableSet.Bytes32Set,uint256) (SpaceDoge.sol#875-877) is never used and should be removed
EnumerableSet.contains(EnumerableSet.AddressSet,address) (SpaceDoge.sol#908-910) is never used and should be removed
EnumerableSet.contains(EnumerableSet.Bytes32Set,bytes32) (SpaceDoge.sol#854-856) is never used and should be removed
EnumerableSet.contains(EnumerableSet.UintSet,uint256) (SpaceDoge.sol#963-965) is never used and should be removed
EnumerableSet.length(EnumerableSet.AddressSet) (SpaceDoge.sol#915-917) is never used and should be removed
EnumerableSet.length(EnumerableSet.Bytes32Set) (SpaceDoge.sol#861-863) is never used and should be removed
EnumerableSet.remove(EnumerableSet.AddressSet,address) (SpaceDoge.sol#901-903) is never used and should be removed
EnumerableSet.remove(EnumerableSet.Bytes32Set,bytes32) (SpaceDoge.sol#847-849) is never used and should be removed
SafeMath.div(uint256,uint256,string) (SpaceDoge.sol#479-482) is never used and should be removed
SafeMath.mod(uint256,uint256) (SpaceDoge.sol#441-444) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (SpaceDoge.sol#499-502) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (SpaceDoge.sol#459-462) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (SpaceDoge.sol#313-317) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (SpaceDoge.sol#349-352) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (SpaceDoge.sol#359-362) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (SpaceDoge.sol#334-342) is never used and should be removed
SafeMath.trySub(uint256,uint256) (SpaceDoge.sol#324-327) 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 (SpaceDoge.sol#3) 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) (SpaceDoge.sol#554-560):
        - (success) = recipient.call{value: amount}() (SpaceDoge.sol#558)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (SpaceDoge.sol#615-622):
        - (success,returndata) = target.call{value: value}(data) (SpaceDoge.sol#620)
Low level call in Address.functionStaticCall(address,bytes,string) (SpaceDoge.sol#640-646):
        - (success,returndata) = target.staticcall(data) (SpaceDoge.sol#644)
Low level call in Address.functionDelegateCall(address,bytes,string) (SpaceDoge.sol#664-670):
        - (success,returndata) = target.delegatecall(data) (SpaceDoge.sol#668)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```


Email: audit@EtherAuthority.io

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ERC721.safeTransferFrom(address,address,uint256,bytes)._data (SpaceDoge.sol#1515) is not in mixedCase
Parameter SpaceDoge.tokensOfOwner(address)._owner (SpaceDoge.sol#1849) is not in mixedCase
Parameter SpaceDoge.exists(uint256)._tokenId (SpaceDoge.sol#1865) is not in mixedCase
Parameter SpaceDoge.setMintPrice(uint256)._price (SpaceDoge.sol#1872) is not in mixedCase
Parameter SpaceDoge.setMaxToMint(uint256)._maxValue (SpaceDoge.sol#1879) is not in mixedCase
Parameter SpaceDoge.reserveDoges(address,uint256)._to (SpaceDoge.sol#1886) is not in mixedCase
Parameter SpaceDoge.reserveDoges(address,uint256)._numberOfTokens (SpaceDoge.sol#1886) is not in mixedCase
Parameter SpaceDoge.setRevealTimestamp(uint256)._revealTimeStamp (SpaceDoge.sol#1899) is not in mixedCase
Parameter SpaceDoge.setProvenanceHash(string)._provenanceHash (SpaceDoge.sol#1906) is not in mixedCase
Variable SpaceDoge.MAX_DOGE_SUPPLY (SpaceDoge.sol#1827) is not in mixedCase
Variable SpaceDoge.REVEAL_TIMESTAMP (SpaceDoge.sol#1828) is not in mixedCase
Variable SpaceDoge.PROVENANCE_HASH (SpaceDoge.sol#1830) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (SpaceDoge.sol#21)" inContext (SpaceDoge.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable SpaceDoge.PROVENANCE_HASH (SpaceDoge.sol#1830) is too similar to SpaceDoge.setProvenanceHash(string)._provenanceHash (SpaceDoge.
sol#1906)
Variable SpaceDoge.REVEAL_TIMESTAMP (SpaceDoge.sol#1828) is too similar to SpaceDoge.setRevealTimestamp(uint256)._revealTimeStamp (SpaceD
oge.sol#1899)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
supportsInterface(bytes4) should be declared external:
        - ERC165.supportsInterface(bytes4) (SpaceDoge.sol#271-273)
name() should be declared external:
        - ERC721.name() (SpaceDoge.sol#1391-1393)
symbol() should be declared external:
        - ERC721.symbol() (SpaceDoge.sol#1398-1400)
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (SpaceDoge.sol#1405-1421)
tokenByIndex(uint256) should be declared external:
        - ERC721.tokenByIndex(uint256) (SpaceDoge.sol#1450-1453)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (SpaceDoge.sol#1458-1467)
setApprovalForAll(address,bool) should be declared external:
```

```
tokenByIndex(uint256) should be declared external:
        - ERC721.tokenByIndex(uint256) (SpaceDoge.sol#1450-1453)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (SpaceDoge.sol#1458-1467)
setApprovalForAll(address,bool) should be declared external:
        - ERC721.setApprovalForAll(address,bool) (SpaceDoge.sol#1481-1486)
transferFrom(address,address,uint256) should be declared external:
        - ERC721.transferFrom(address,address,uint256) (SpaceDoge.sol#1498-1503)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721.safeTransferFrom(address,address,uint256) (SpaceDoge.sol#1508-1510)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (SpaceDoge.sol#1799-1802)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (SpaceDoge.sol#1808-1812)
exists(uint256) should be declared external:
        - SpaceDoge.exists(uint256) (SpaceDoge.sol#1865-1867)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:SpaceDoge.sol analyzed (15 contracts with 75 detectors), 88 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity static analysis

SpaceDoge.sol

Email: audit@EtherAuthority.io

Selfdestruct:

INTERNAL ERROR in module Selfdestruct: can't convert undefined to object

Pos: not available

Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: can't convert undefined to object

Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: can't convert undefined to object

Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: can't convert undefined to object

Pos: not available

Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: can't convert undefined to object

Pos: not available

ERC

ERC20:

INTERNAL ERROR in module ERC20: can't convert undefined to object

Pos: not available

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: can't convert undefined to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: can't convert undefined to object
Pos: not available

No return:

INTERNAL ERROR in module No return: can't convert undefined to object
Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: can't convert undefined to object
Pos: not available

String length:

INTERNAL ERROR in module String length: can't convert undefined to object
Pos: not available

Solhint Linter

SpaceDoge.sol

contracts/SpaceDoge.sol:3:1: Error: Compiler version ^0.7.6 does not satisfy the r semver requirement

contracts/SpaceDoge.sol:215:1: Error: Compiler version >=0.6.0 <0.8.0 does not satisfy the r semver requirement

contracts/SpaceDoge.sol:253:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)

contracts/SpaceDoge.sol:286:1: Error: Compiler version >=0.6.0 <0.8.0 does not satisfy the r semver requirement

contracts/SpaceDoge.sol:687:1: Error: Compiler version >=0.6.0 <0.8.0 does not satisfy the r semver requirement

contracts/SpaceDoge.sol:987:1: Error: Compiler version >=0.6.0 <0.8.0 does not satisfy the r semver requirement

```
contracts/SpaceDoge.sol:1361:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)

contracts/SpaceDoge.sol:1745:95: Error: Code contains empty blocks

contracts/SpaceDoge.sol:1769:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)

contracts/SpaceDoge.sol:1825:20: Error: Variable name must be in mixedCase

contracts/SpaceDoge.sol:1826:20: Error: Variable name must be in mixedCase

contracts/SpaceDoge.sol:1828:19: Error: Variable name must be in mixedCase

contracts/SpaceDoge.sol:1831:5: Error: Explicitly mark visibility of state

contracts/SpaceDoge.sol:1832:5: Error: Explicitly mark visibility of state

contracts/SpaceDoge.sol:1834:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)

contracts/SpaceDoge.sol:1936:77: Error: Avoid to make time-based decisions in your business logic
```

Email: audit@EtherAuthority.io