Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Digital Doka
Website:	www.digitaldoka.xyz
Platform:	Ethereum
Language:	Solidity
Date:	April 25th, 2023

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 Doka Digital team to perform the Security audit of the Doka Digital 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 April 25th, 2023.

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

- Doka Digital is a web3 project by Studio 9 Labs and Phantasm Labs with the aim of revolutionizing anime through the use of blockchain, storytelling and the team's decades of experience with anime and finance.
- The Doka Digital is an ERC721A token which has withdraw, airdrop, stopMint functionalities.
- Doka Digital contract inherits Ownable, ERC2981, ECDSA standard smart contracts from the OpenZeppelin library.
- Doka Digital contract inherits OperatorFilterer standard smart contracts from the Vectorized from github library and ERC721A standard smart contracts from the chiru-labs from github 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 Doka Digital Smart Contract**
Platform	**Ethereum / Solidity**
File	Doka.sol
File MD5 Hash	EA95C6235875E5A96E3C0B62898C1FA3
Audit Date	April 25th, 2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Doka • Symbol: DOKA • Decimals: 18	**YES, This is valid.**

• Initial Royalty: 5%
• Reserve Price: 0.088 ether
• Maximum Supply: 5555
  ○ 100 reserved for the team.
  ○ 5455 available for the reservation.
	YES, This is valid.

• Here 2 phases are used.
  ○ Phase 1 (WL)
  ■ Both public & wl users can reserve tokens (2 max each wallet).
  ■ WL has priority, capped at 5455.
  ■ Public can only reserve if there are available spots left.
  ■ E.g. if 5055 are reserved by WL, only 400 can be reserved by the public.
  ○ Phase 2 (public)
  ■ Reservation open to the public.
  ■ Public can only reserve if there are available spots left.
	YES, This is valid.

Owner Specifications: • Owner calls airdrop after the reservations phases are finished to distribute tokens. • Owner calls to withdraw funds from the contract. • Owner calls stopMint to stop minting forever. • Current owner can transfer ownership of the contract to	**YES, This is valid.**

a new account. ● Deleting ownership will leave the contract without an owner, removing any owner-only functionality.	

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.
We confirm that 2 informational issues are 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	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	Passed
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	Moderated
	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 Doka 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 Doka Token.

The Doka 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 in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given a Doka 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 **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://www.digitaldoka.xyz 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 not 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	_nextTokenId	internal	Passed	No Issue
4	totalSupply	read	Passed	No Issue
5	_totalMinted	internal	Passed	No Issue
6	_totalBurned	internal	Passed	No Issue
7	balanceOf	read	Passed	No Issue
8	_numberMinted	internal	Passed	No Issue
9	_numberBurned	internal	Passed	No Issue
10	_getAux	internal	Passed	No Issue
11	_setAux	internal	Passed	No Issue
12	supportsInterface	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	ownerOf	read	Passed	No Issue
18	_ownershipOf	internal	Passed	No Issue
19	_ownershipAt	internal	Passed	No Issue
20	_ownershipIsInitialized	internal	Passed	No Issue
21	_initializeOwnershipAt	internal	Passed	No Issue
22	_packedOwnershipOf	read	Passed	No Issue
23	_unpackedOwnership	write	Passed	No Issue
24	_packOwnershipData	read	Passed	No Issue
25	_nextInitializedFlag	write	Passed	No Issue
26	approve	write	Passed	No Issue
27	getApproved	read	Passed	No Issue
28	setApprovalForAll	write	Passed	No Issue
29	isApprovedForAll	read	Passed	No Issue
30	_exists	internal	Passed	No Issue
31	_isSenderApprovedOrOwner	write	Passed	No Issue
32	_getApprovedSlotAndAddress	read	Passed	No Issue
33	transferFrom	write	Passed	No Issue
34	safeTransferFrom	write	Passed	No Issue
35	safeTransferFrom	write	Passed	No Issue
36	_beforeTokenTransfers	internal	Passed	No Issue
37	_afterTokenTransfers	internal	Passed	No Issue
38	_checkContractOnERC721Received	write	Passed	No Issue
39	_mint	internal	Passed	No Issue
40	_mintERC2309	internal	Passed	No Issue
41	_safeMint	internal	Passed	No Issue

42	_safeMint	internal	Passed	No Issue
43	_approve	internal	Passed	No Issue
44	_approve	internal	Passed	No Issue
45	_burn	internal	Passed	No Issue
46	_burn	internal	Passed	No Issue
47	_setExtraDataAt	internal	Passed	No Issue
48	_extraData	internal	Passed	No Issue
49	_nextExtraData	read	Passed	No Issue
50	_msgSenderERC721A	internal	Passed	No Issue
51	_toString	internal	Passed	No Issue
52	_revert	internal	Passed	No Issue
53	onlyOwner	modifier	Passed	No Issue
54	owner	read	Passed	No Issue
55	_checkOwner	internal	Passed	No Issue
56	renounceOwnership	write	access only Owner	No Issue
57	transferOwnership	write	access only Owner	No Issue
58	_transferOwnership	internal	Passed	No Issue
59	supportsInterface	read	Passed	No Issue
60	royaltyInfo	read	Passed	No Issue
61	_feeDenominator	internal	Passed	No Issue
62	_setDefaultRoyalty	internal	Passed	No Issue
63	_deleteDefaultRoyalty	internal	Passed	No Issue
64	_setTokenRoyalty	internal	Passed	No Issue
65	_resetTokenRoyalty	internal	Passed	No Issue
66	_registerForOperatorFiltering	internal	Passed	No Issue
67	_registerForOperatorFiltering	internal	Passed	No Issue
68	onlyAllowedOperator	modifier	Passed	No Issue
69	onlyAllowedOperatorApproval	modifier	Passed	No Issue
70	_revertIfBlocked	read	Passed	No Issue
71	_operatorFilteringEnabled	internal	Passed	No Issue
72	_isPriorityOperator	internal	Passed	No Issue
73	setApprovalForAll	write	access only Allowed Operator Approval	No Issue
74	approve	write	access only Allowed Operator Approval	No Issue
75	transferFrom	write	access only Allowed Operator	No Issue
76	_isPriorityOperator	internal	Passed	No Issue
77	setDefaultRoyalty	write	access only Owner	No Issue
78	deleteDefaultRoyalty	write	access only Owner	No Issue
79	supportsInterface	read	Passed	No Issue
80	_startTokenId	internal	Passed	No Issue
81	_baseURI	internal	Passed	No Issue
82	tokenURI	read	Passed	No Issue
83	reserve	external	Passed	No Issue
84	setSigner	external	access only Owner	No Issue
85	setOperatorFilteringEnabled	write	access only Owner	No Issue
86	setPhase	external	access only Owner	No Issue

87	setReservePrice	external	access only Owner	No Issue
88	setBaseURI	write	Passed	No Issue
89	withdraw	external	Owner can withdraw all funds, Transferred 0 amount	Refer to audit findings
90	airdrop	external	Infinite loops possibility	Refer to audit findings
91	stopMint	external	access only Owner	No Issue
92	signer	external	Passed	No Issue
93	operatorFilteringEnabled	external	Passed	No Issue
94	phase	external	Passed	No Issue
95	reservePrice	external	Passed	No Issue
96	baseURI	external	Passed	No Issue
97	totalReserveCounter	external	Passed	No Issue
98	wlReserveCounter	external	Passed	No Issue
99	publicReserveCounter	external	Passed	No Issue
100	wlReserveAddresses	external	Passed	No Issue
101	publicReserveAddresses	external	Passed	No Issue
102	reserveCounter	external	Passed	No Issue
103	mintEnabled	external	Passed	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.

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) Infinite loops possibility:

As array elements will increase, then it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop. We suggest avoiding loops. For example, use mapping to store the array index. And query that data directly, instead of looping through all the elements to find an element.

Resolution: Adjust logic to replace loops with mapping or other code structure.
airdrop() - holders_.length.

(2) Owner can withdraw all funds:

```
// withdraw
function withdraw() external onlyOwner {
    uint256 balance = address(this).balance;

    address _wallet = TEAM_ADDRESS;
    uint256 _payable = balance;
    payable(_wallet).transfer(_payable);
}
```

There is a withdraw() function, with a restriction to call this function only owner, And the owner can withdraw all the funds of the smart contract.

Resolution: We suggest cross verify logic for withdrawing funds by the owner. If this is a part of the plan then disregard this issue.

Status: This is acknowledged in the revised smart contract code.

(3) Transferred 0 amount:

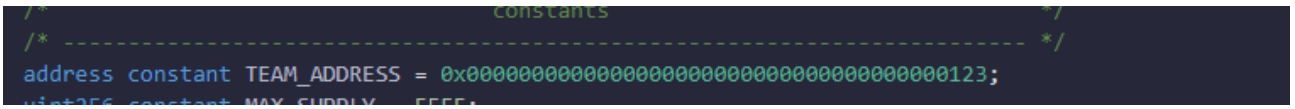
```
// withdraw
function withdraw() external onlyOwner {
    uint256 balance = address(this).balance;

    address _wallet = TEAM_ADDRESS;
    uint256 _payable = balance;
    payable(_wallet).transfer(_payable);
}
```

Withdraw function, there is no checks contract balance is not zero, before transfer.
Transfers 0 amounts.

Resolution: We suggest avoiding 0 amounts to get transferred to an Unused Internal function.

Status: This is acknowledged in the revised smart contract code.

(4) Hardcoded value:



TEAM_ADDRESS is defined as constant and set by the address.

Resolution: We suggest confirming this address before deploying the smart contract. This address is used to withdraw coins from the contract.

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:

Doka.sol

- setDefaultRoyalty: Royalty address can be set by the owner.
- deleteDefaultRoyalty: Royalty address can be deleted by the owner.
- setSigner: Signer address can be set by the owner.
- setOperatorFilteringEnabled: Operator Filtering Enabled value can be set by the owner.
- setPhase: Phase can be set by the owner.
- setReservePrice: Reserve price can be set by the owner.
- setBaseURI: BaseURI can be set by the owner.
- withdraw: Owner calls to withdraw funds from the contract.
- airdrop: Owner calls airdrop after the reservations phases are finished to distribute tokens.
- stopMint: Owner calls stopMint to stop minting forever.

Ownable.sol

- renounceOwnership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.
- _checkOwner: Check if the sender is not the owner then throws.

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 observed some informational severity issues in the token smart contract. We confirm that there are no major issues in the revised smart contract code. 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).

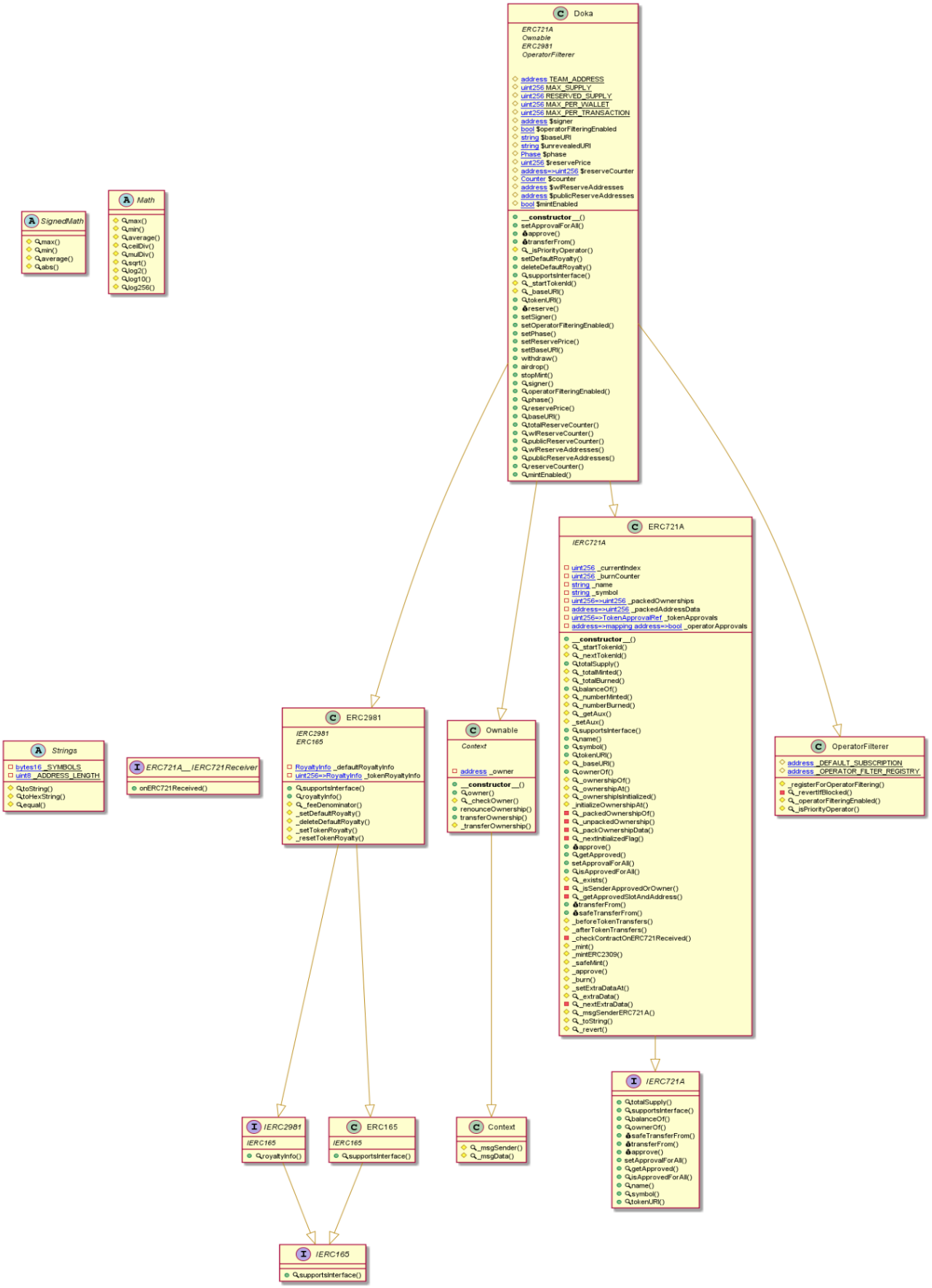
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither Log >> Doka.sol

```
Doka.setReservePrice(uint256) (Doka.sol#1360-1362) should emit an event for:
        - $reservePrice = reservePrice_ (Doka.sol#1361)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Doka.constructor(address).signer_ (Doka.sol#1250) lacks a zero-check on :
        - $signer = signer_ (Doka.sol#1251)
Doka.setSigner(address).signer_ (Doka.sol#1348) lacks a zero-check on :
        - $signer = signer_ (Doka.sol#1349)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).retval (Doka.sol#919)' in ERC721A._checkContra
ctOnERC721Received(address,address,uint256,bytes) (Doka.sol#912-929) potentially used before declaration: retval == ERC721A__I
ERC721Receiver(to).onERC721Received.selector (Doka.sol#921)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).reason (Doka.sol#922)' in ERC721A._checkContra
ctOnERC721Received(address,address,uint256,bytes) (Doka.sol#912-929) potentially used before declaration: reason.length == 0 (
Doka.sol#923)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).reason (Doka.sol#922)' in ERC721A._checkContra
ctOnERC721Received(address,address,uint256,bytes) (Doka.sol#912-929) potentially used before declaration: revert(uint256,uint2
56)(32 + reason,mload(uint256)(reason)) (Doka.sol#926)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Math.mulDiv(uint256,uint256,uint256) (Doka.sol#48-97) uses assembly
        - INLINE ASM (Doka.sol#52-56)
        - INLINE ASM (Doka.sol#66-71)
        - INLINE ASM (Doka.sol#75-81)
Strings.toString(uint256) (Doka.sol#256-274) uses assembly
        - INLINE ASM (Doka.sol#261-263)
        - INLINE ASM (Doka.sol#266-268)
ECDSA.tryRecover(bytes32,bytes) (Doka.sol#327-341) uses assembly
        - INLINE ASM (Doka.sol#332-336)
ECDSA.toEthSignedMessageHash(bytes32) (Doka.sol#380-386) uses assembly
        - INLINE ASM (Doka.sol#381-385)
ECDSA.toTypedDataHash(bytes32,bytes32) (Doka.sol#392-400) uses assembly
        - INLINE ASM (Doka.sol#393-399)
ERC721A._setAux(address,uint64) (Doka.sol#690-698) uses assembly
        - INLINE ASM (Doka.sol#693-695)
```

```
ERC721A._revert(bytes4) (Doka.sol#1121-1126) uses assembly
        - INLINE ASM (Doka.sol#1122-1125)
OperatorFilterer._registerForOperatorFiltering(address,bool) (Doka.sol#1137-1164) uses assembly
        - INLINE ASM (Doka.sol#1141-1163)
OperatorFilterer._revertIfBlocked(address) (Doka.sol#1182-1196) uses assembly
        - INLINE ASM (Doka.sol#1183-1195)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ERC721A._mint(address,uint256) (Doka.sol#932-954) has costly operations inside a loop:
        - _currentIndex = end (Doka.sol#951)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
Context._msgData() (Doka.sol#549-551) is never used and should be removed
ECDSA._throwError(ECDSA.RecoverError) (Doka.sol#315-325) is never used and should be removed
ECDSA.recover(bytes32,bytes) (Doka.sol#343-347) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (Doka.sol#355-359) is never used and should be removed
ECDSA.recover(bytes32,uint8,bytes32,bytes32) (Doka.sol#374-378) is never used and should be removed
ECDSA.toDataWithIntendedValidatorHash(address,bytes) (Doka.sol#402-404) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes) (Doka.sol#388-390) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes32) (Doka.sol#380-386) is never used and should be removed
ECDSA.toTypedDataHash(bytes32,bytes32) (Doka.sol#392-400) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes) (Doka.sol#327-341) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes32,bytes32) (Doka.sol#349-353) is never used and should be removed
ECDSA.tryRecover(bytes32,uint8,bytes32,bytes32) (Doka.sol#361-372) is never used and should be removed
ERC2981._resetTokenRoyalty(uint256) (Doka.sol#540-542) is never used and should be removed
ERC2981._setTokenRoyalty(uint256,address,uint96) (Doka.sol#533-538) is never used and should be removed
ERC721A._approve(address,uint256) (Doka.sol#1001-1003) is never used and should be removed
ERC721A._baseURI() (Doka.sol#723-725) is never used and should be removed
ERC721A._burn(uint256) (Doka.sol#1021-1023) is never used and should be removed
ERC721A._burn(uint256,bool) (Doka.sol#1025-1066) is never used and should be removed
ERC721A._exists(uint256) (Doka.sol#800-808) is never used and should be removed
ERC721A._getAux(address) (Doka.sol#686-688) is never used and should be removed
ERC721A._initializeOwnershipAt(uint256) (Doka.sol#744-748) is never used and should be removed
ERC721A._mintERC2309(address,uint256) (Doka.sol#956-974) is never used and should be removed
ERC721A._nextTokenId() (Doka.sol#653-655) is never used and should be removed
ERC721A._numberBurned(address) (Doka.sol#682-684) is never used and should be removed
ERC721A._numberMinted(address) (Doka.sol#678-680) is never used and should be removed
ERC721A._ownershipAt(uint256) (Doka.sol#736-738) is never used and should be removed
ERC721A._ownershipIsInitialized(uint256) (Doka.sol#740-742) is never used and should be removed
ERC721A._ownershipOf(uint256) (Doka.sol#732-734) is never used and should be removed
ERC721A._revert(bytes4) (Doka.sol#1121-1126) is never used and should be removed
ERC721A._safeMint(address,uint256) (Doka.sol#996-998) is never used and should be removed
ERC721A._safeMint(address,uint256,bytes) (Doka.sol#976-994) is never used and should be removed
```

```
Math.sqrt(uint256,Math.Rounding) (Doka.sol#126-131) is never used and should be removed
OperatorFilterer._isPriorityOperator(address) (Doka.sol#1202-1204) is never used and should be removed
SignedMath.abs(int256) (Doka.sol#19-23) is never used and should be removed
SignedMath.average(int256,int256) (Doka.sol#14-17) is never used and should be removed
SignedMath.max(int256,int256) (Doka.sol#6-8) is never used and should be removed
SignedMath.min(int256,int256) (Doka.sol#10-12) is never used and should be removed
Strings.equal(string,string) (Doka.sol#302-304) is never used and should be removed
Strings.toHexString(address) (Doka.sol#298-300) is never used and should be removed
Strings.toHexString(uint256) (Doka.sol#280-284) is never used and should be removed
Strings.toHexString(uint256,uint256) (Doka.sol#286-296) is never used and should be removed
Strings.toString(int256) (Doka.sol#276-278) is never used and should be removed
Strings.toString(uint256) (Doka.sol#256-274) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.17 (Doka.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.1
6
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Contract ERC721A__IERC721Receiver (Doka.sol#470-477) is not in CapWords
Parameter ERC721A.safeTransferFrom(address,address,uint256,bytes)._data (Doka.sol#890) is not in mixedCase
Variable Doka.$signer (Doka.sol#1218) is not in mixedCase
Variable Doka.$operatorFilteringEnabled (Doka.sol#1220) is not in mixedCase
Variable Doka.$baseURI (Doka.sol#1222) is not in mixedCase
Variable Doka.$unrevealedURI (Doka.sol#1223) is not in mixedCase
Variable Doka.$phase (Doka.sol#1231) is not in mixedCase
Variable Doka.$reservePrice (Doka.sol#1233) is not in mixedCase
Variable Doka.$reserveCounter (Doka.sol#1235) is not in mixedCase
Variable Doka.$counter (Doka.sol#1243) is not in mixedCase
Variable Doka.$wlReserveAddresses (Doka.sol#1245) is not in mixedCase
Variable Doka.$publicReserveAddresses (Doka.sol#1246) is not in mixedCase
Variable Doka.$mintEnabled (Doka.sol#1248) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable Doka.$reservePrice (Doka.sol#1233) is too similar to Doka.setReservePrice(uint256).reservePrice_ (Doka.sol#1360)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

Doka.RESERVED_SUPPLY (Doka.sol#1214) is never used in Doka (Doka.sol#1207-1448)
Doka.MAX_PER_WALLET (Doka.sol#1215) is never used in Doka (Doka.sol#1207-1448)
Doka.MAX_PER_TRANSACTION (Doka.sol#1216) is never used in Doka (Doka.sol#1207-1448)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

```
Doka.$unrevealedURI (Doka.sol#1223) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
Doka.sol analyzed (15 contracts with 84 detectors), 116 result(s) found
```

Solidity Static Analysis

Doka.sol

Security

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 2014:12:

Gas & Economy

Gas costs:

Gas requirement of function Doka.reserve 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: 2209:8:

Gas costs:

Gas requirement of function Doka.airdrop 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: 2308:8:

Email: audit@EtherAuthority.io

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: 2313:12:

Miscellaneous

Constant/View/Pure functions:

Doka.setApprovalForAll(address,bool) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 2127:8:

Similar variable names:

Doka.setBaseURI(string) : Variables have very similar names "$baseURI" and "baseURI_". Note: Modifiers are currently not considered by this static analysis.

Pos: 2290:23:

Similar variable names:

Doka.setBaseURI(string) : Variables have very similar names "$baseURI" and "baseURI_". Note: Modifiers are currently not considered by this static analysis.

Pos: 2290:23:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 754:12:

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: 699:36:

Solhint Linter

Doka.sol

```
Doka.sol:34:18: Error: Parse error: missing ';' at '{'
Doka.sol:87:18: Error: Parse error: missing ';' at '{'
Doka.sol:204:18: Error: Parse error: missing ';' at '{'
Doka.sol:220:18: Error: Parse error: missing ';' at '{'
Doka.sol:232:18: Error: Parse error: missing ';' at '{'
Doka.sol:273:18: Error: Parse error: missing ';' at '{'
Doka.sol:285:18: Error: Parse error: missing ';' at '{'
Doka.sol:322:18: Error: Parse error: missing ';' at '{'
Doka.sol:336:18: Error: Parse error: missing ';' at '{'
Doka.sol:365:18: Error: Parse error: missing ';' at '{'
Doka.sol:379:18: Error: Parse error: missing ';' at '{'
Doka.sol:411:18: Error: Parse error: missing ';' at '{'
Doka.sol:547:43: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:549:42: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:551:36: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:553:27: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:555:26: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:557:39: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:559:43: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:561:36: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:563:48: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:565:31: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:567:37: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:569:41: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:571:45: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:982:18: Error: Parse error: missing ';' at '{'
Doka.sol:993:18: Error: Parse error: missing ';' at '{'
Doka.sol:1179:30: Error: Parse error: missing ';' at '{'
Doka.sol:1384:18: Error: Parse error: missing ';' at '{'
Doka.sol:1569:18: Error: Parse error: missing ';' at '{'
Doka.sol:1646:18: Error: Parse error: missing ';' at '{'
Doka.sol:1691:18: Error: Parse error: missing ';' at '{'
Doka.sol:1800:18: Error: Parse error: missing ';' at '{'
Doka.sol:1833:48: Error: Parse error: mismatched input ';' expecting
'('
Doka.sol:1837:18: Error: Parse error: missing ';' at '{'
```

```
Doka.sol:2042:25: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2043:26: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2044:23: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2045:21: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2046:32: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2047:37: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2048:26: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2049:29: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2050:25: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2211:44: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2212:77: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2222:60: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2223:63: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2224:73: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2225:68: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2232:90: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2240:85: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2244:88: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2249:79: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2310:34: Error: Parse error: mismatched input '(' expecting
{';', '='}
Doka.sol:2316:22: Error: Parse error: missing ';' at '{'
Doka.sol:2322:35: Error: Parse error: mismatched input '(' expecting
{';', '='}
```

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