SOLIDProof

Blockchain Security | Smart Contract Audits

MADE IN GERMANY

Audit Passed

Security Assessment

8. June, 2021

For

KOJI

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof’s position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Overview

Network
Ethereum (ERC20)

Website
https://koji.earth

Telegram
https://t.me/kojiearth

Twitter
https://twitter.com/kojiearth

Facebook
https://facebook.com/thekojiearth

Github
https://github.com/nodezy/kojiearth

Reddit
https://www.reddit.com/r/kojiearth

Description

koji.earth is an ERC-20 project on the Ethereum network. A community driven token, created to help those in need via mutual aid and donations from 1% of each transaction, brought to earth by Koji, an alien with the core mission of helping the earth in times of crisis by cooperating with charitable organizations. In simple terms, KOJI is a hybrid digital token: a DeFi Charity following a deflationary model with redistribution features and regular NFT drops. They aim to cement their position as the leading mutual-aid token by helping the world while offer best possible setting for a great ROI. Deflationary and rewarding by design with 0.5% KOJI burned + 1% redistribution back to all holders from each transaction made including regular NFT drops. Speaking of the NFT drops; they recently signed a deal with the professional comic & grapich novel company AmCo Studios to produce the Koji Comics & The Kojiverse that will be used for their exclusive and first-of-its kind NFT Comics.

Project Engagement

During the 6th of June, **Koji Token Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. **Koji Token Team** provided Solidproof.io with access to their code repository and whitepaper.

Logo

Modified: From white to black (https://koji.earth/wp-core/wp-content/themes/koji.earth/assets/imgs/koji-type.svg)

Contract Link

https://etherscan.io/address/0x1c8266a4369af6d80df2659ba47b3c98f35cb8be#code

Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    i) Review of the specifications, sources, and instructions provided to SolidProof to make sure we understand the size, scope, and functionality of the smart contract.
    ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to SolidProof describe.

2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

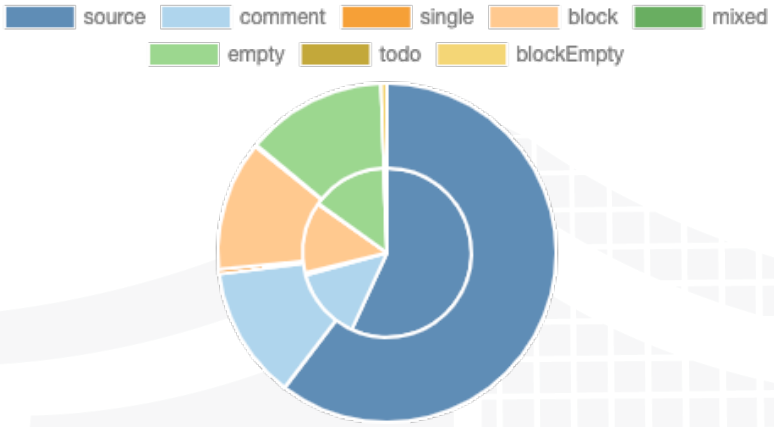
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

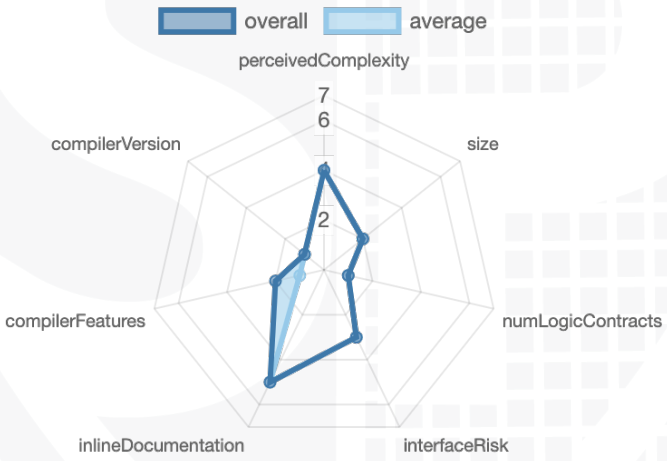
Dependency / Import Path	Count
@openzeppelin/contracts/GSN/Context.sol	1
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/math/SafeMath.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/utils/Address.sol	1

Metrics

Source Lines



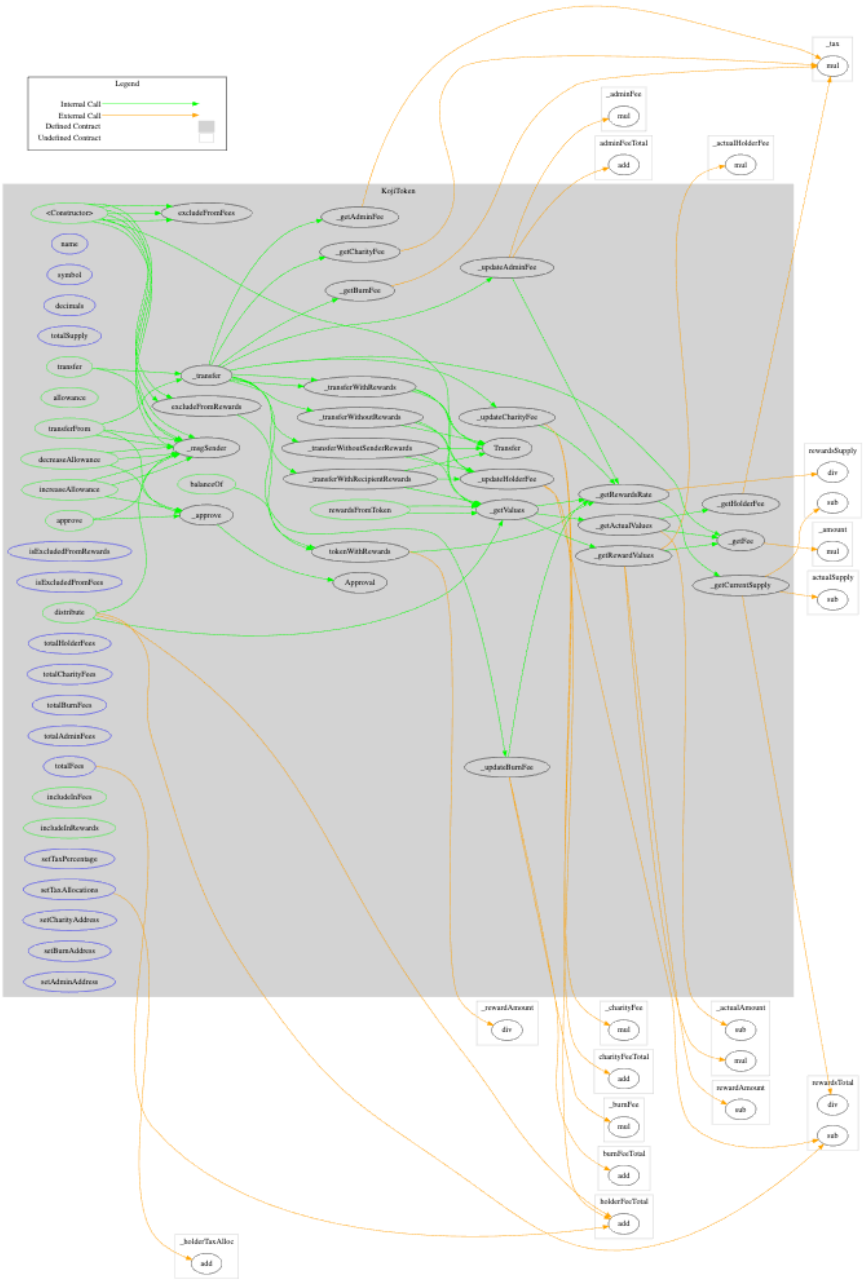
Risk Level



Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
0.7.4			**** (0 asm blocks)	

CallGraph



Source Units in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/Kojitoken.sol	1	———	617	554	373	91	285	———
📝	**Totals**	**1**	———	**617**	**554**	**373**	**91**	**285**	———

Audit Results

AUDIT PASSED

Critical issues

- no critical issues found -

High issues

- no hight issues found -

Medium issues

- no medium issues found -

Low issues

Issue	File	Description	Line
_charityAdress	Main	Lacks of Zero.	140: charityAddress = _charityAddress
_adminAdress	Main	Lacks of Zero.	614: adminAddress = _adminAddress
_burnAdress	Main	Lacks of Zero.	610: burnAddress = _burnAddress

Meanings:

Lacks of Zero:

- There exists Zero address in solidity e.g. adress(0) or adress(0x0)
- In Solidity 0.5.0 comparing with integer 0 is not allowed anymore

Informational issues

Issue	File	Description	Line
#1	Main	Used literals with too many digits	150: excludeFromFees(address(0x0000000000000000000000000000dEaD));
#2	Main	not in mixedCase	169: function balanceOf(address _account) public view override returns (uint256) {
#3	Main	not in mixedCase	181: function allowance(address _owner, address _spender) public view override returns (uint256) {
#4	Main	not in mixedCase	185: function approve(address _spender, uint256 _amount) public override returns (bool) {
#5	Main	not in mixedCase	211: function decreaseAllowance(address _spender, uint256 _subtractedValue) public virtual returns (bool) {
#6	Main	not in mixedCase	475: function rewardsFromToken(uint256 _actualAmount, bool _deductTransferFee) public view returns (uint256) {

SWC Attacks

ID	Title	Relationships	Status
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED

SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

Blockchain Security | Smart Contract Audits

MADE IN GERMANY