SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment
30. November, 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.

Version	Date	Description
1.0	30. November 2021	• Layout project • Automated- /Manual-Security Testing • Summary

Network

Binance Smart Chain (BEP20)

Website

https://koji.earth/

Telegram

https://t.me/kojiearth

Twitter

https://twitter.com/kojiearth

Instagram

https://instagram.com/kojiearth

Reddit

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

Description

koji.earth, initially an ERC20 token, and has now moved to the Binance Smart Chain (BSC) network to help dealing with the increasing gas fees associated with the Ethereum network (read more about the move & reasoning behind it here). KOJI is 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. We aim to cement our 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.

KOJI is a community driven token so make sure you join the Telegram group kojiearth to help steer our direction and make sure to get onboard early. Our website is still under construction, the telegram channels will have the most up-to-date news and announcements.

Project Engagement

During the 24th of November 2021, **KOJI 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. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

v1.0

https://bscscan.com/address/0x99919114a6e249a9d7862422211d37c41ea29589#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.

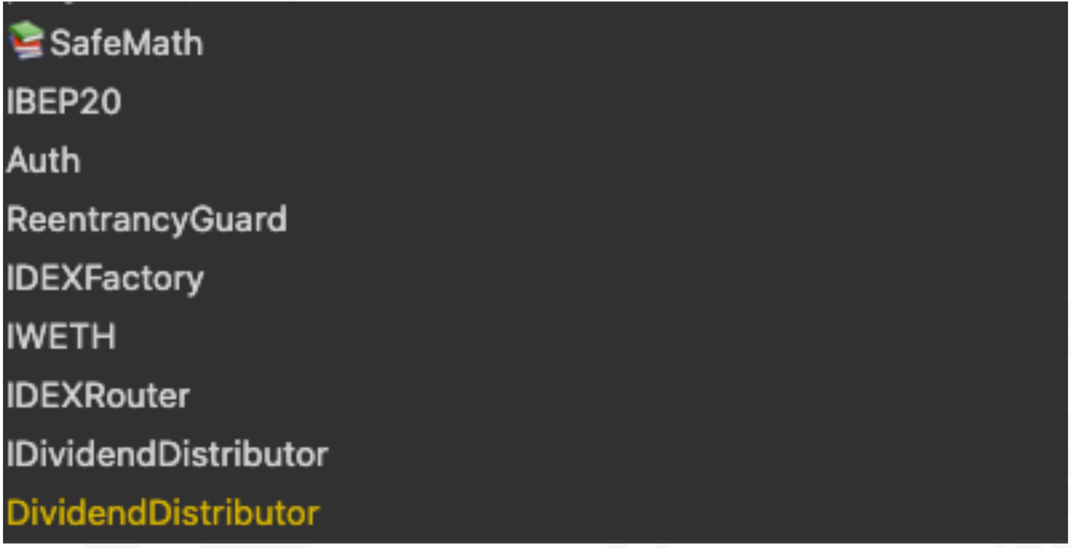
2. Testing and automated analysis that includes the following:
   i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii) Symbolic execution, which is analysing a program to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

Imported packages:



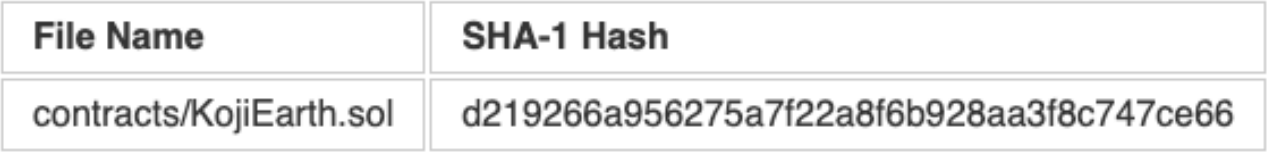
SafeMath
IBEP20
Auth
ReentrancyGuard
IDEXFactory
IWETH
IDEXRouter
IDividendDistributor
DividendDistributor

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

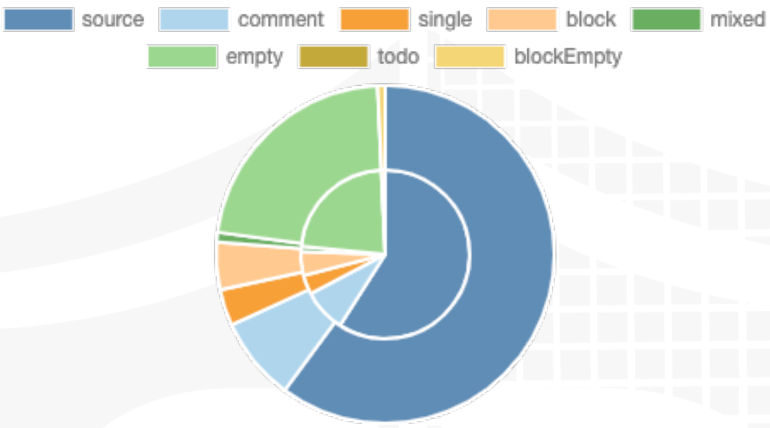


File Name	SHA-1 Hash
contracts/KojiEarth.sol	d219266a956275a7f22a8f6b928aa3f8c747ce66

Metrics

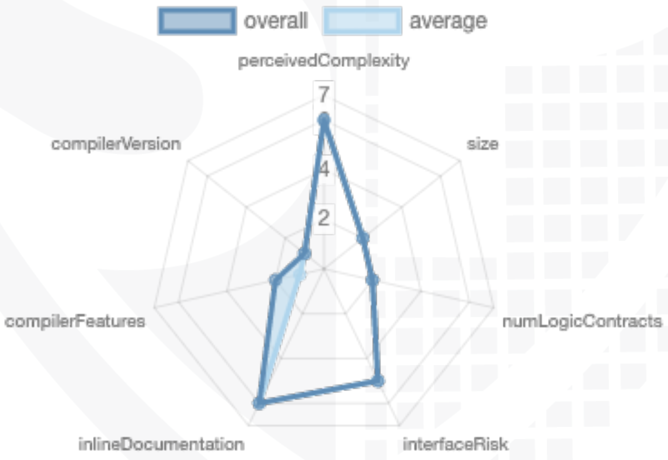
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	1	5	2

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Version	Public	Payable
1.0	119	8

Version	External	Internal	Private	Pure	View
1.0	104	109	0	11	41

State Variables

Version	Total	Public
1.0	84	43

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.8.9`		`yes`	`yes` (1 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2

1.0	yes					yes → New Contract:DividendDistributor

Scope of Work

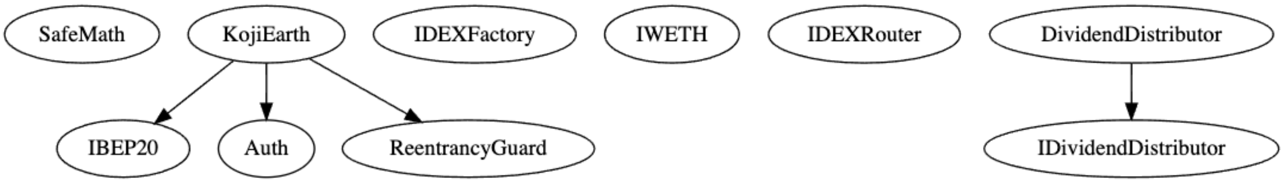
The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Inheritance Graph

v1.0



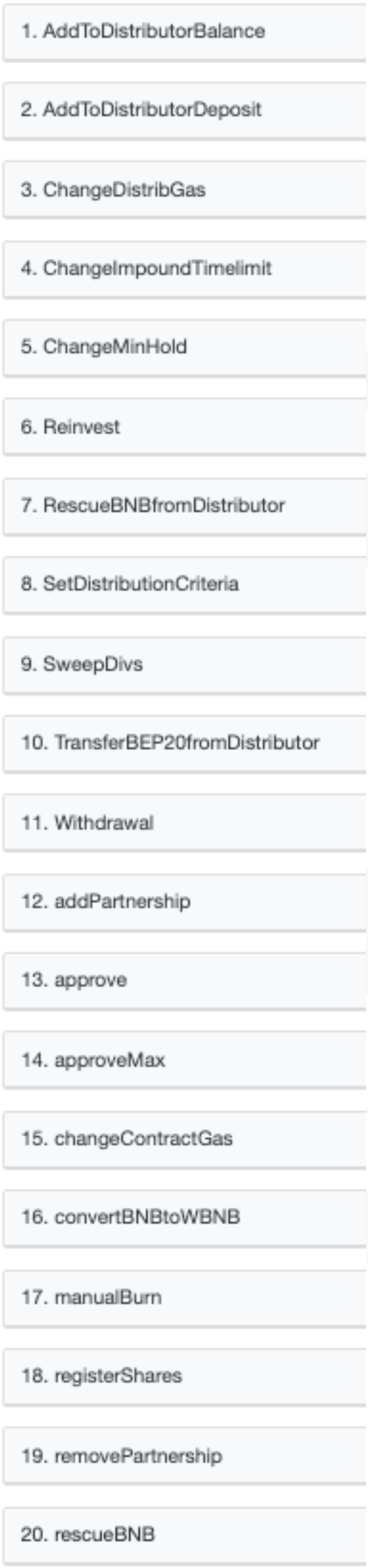
Verify Claims

Correct implementation of Token standard

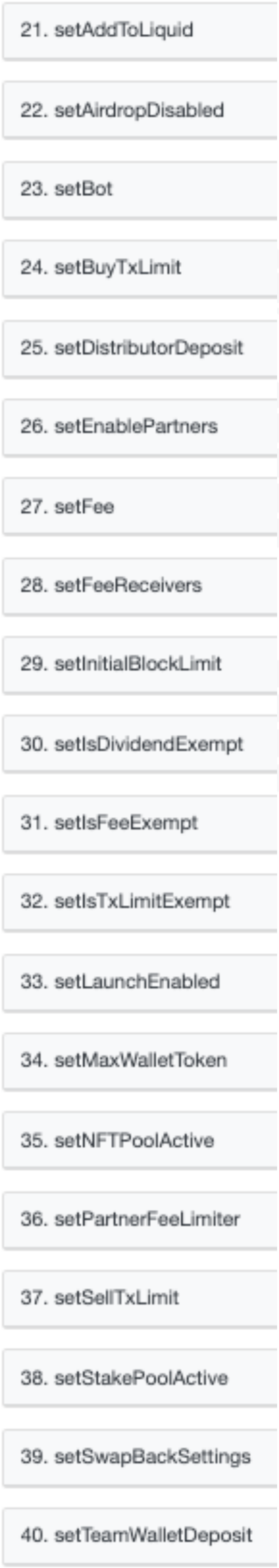
Tested	Verified
✓	✓

Function	Description	Exist	Tested	Verified
TotalSupply	provides information about the total token supply	✓	✓	✓
BalanceOf	provides account balance of the owner's account	✓	✓	✓
Transfer	executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	returns a set number of tokens from a spender to the owner	✓	✓	✓

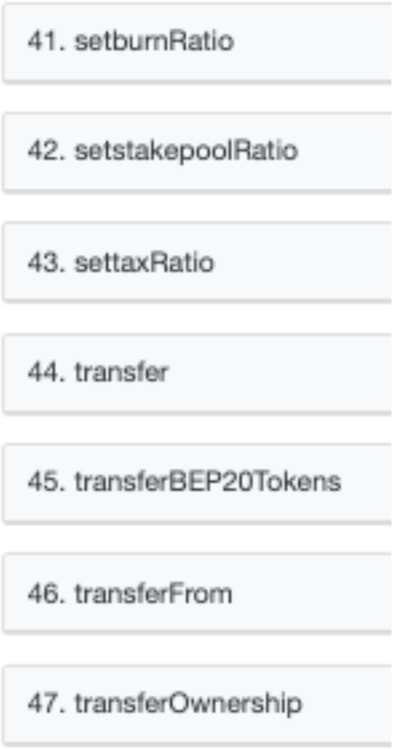
Write functions of contract



1. AddToDistributorBalance
2. AddToDistributorDeposit
3. ChangeDistribGas
4. ChangeImpoundTimelimit
5. ChangeMinHold
6. Reinvest
7. RescueBNBfromDistributor
8. SetDistributionCriteria
9. SweepDivs
10. TransferBEP20fromDistributor
11. Withdrawal
12. addPartnership
13. approve
14. approveMax
15. changeContractGas
16. convertBNBtoWBNB
17. manualBurn
18. registerShares
19. removePartnership
20. rescueBNB



21. setAddToLiquid
22. setAirdropDisabled
23. setBot
24. setBuyTxLimit
25. setDistributorDeposit
26. setEnablePartners
27. setFee
28. setFeeReceivers
29. setInitialBlockLimit
30. setIsDividendExempt
31. setIsFeeExempt
32. setIsTxLimitExempt
33. setLaunchEnabled
34. setMaxWalletToken
35. setNFTPoolActive
36. setPartnerFeeLimiter
37. setSellTxLimit
38. setStakePoolActive
39. setSwapBackSettings
40. setTeamWalletDeposit



41. setburnRatio
42. setstakepoolRatio
43. settaxRatio
44. transfer
45. transferBEP20Tokens
46. transferFrom
47. transferOwnership

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified	File
Deployer cannot mint	✓	✓	✓	Main
Comment	Line: -			

Max / Total Supply: 1.000.000.000.000

Deployer cannot burn or lock user funds

Name	Exist	Tested	Verified
Deployer cannot lock	✓	✓	✗
Deployer cannot burn	✓	✓	✓

Comments:

v1.0

- Deployer can lock user funds
  - If _maxTxAmountBuy is 0

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	–	–	–

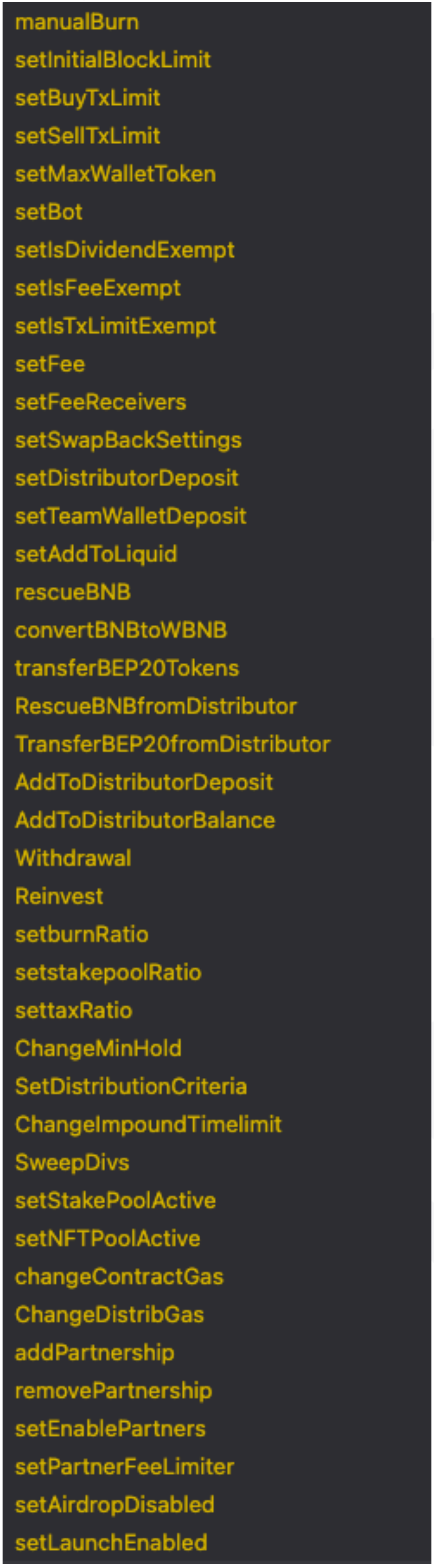
Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verfified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	–

OnlyOwner functions

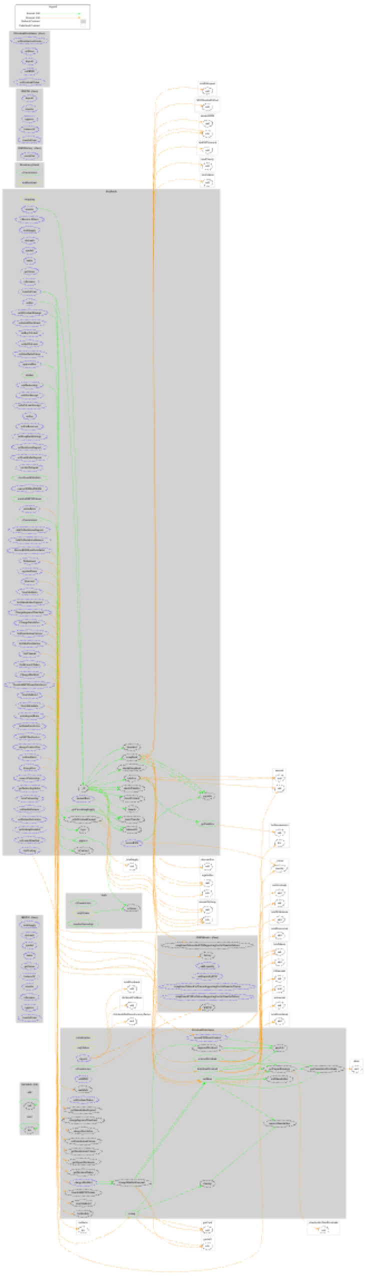


```
manualBurn
setInitialBlockLimit
setBuyTxLimit
setSellTxLimit
setMaxWalletToken
setBot
setIsDividendExempt
setIsFeeExempt
setIsTxLimitExempt
setFee
setFeeReceivers
setSwapBackSettings
setDistributorDeposit
setTeamWalletDeposit
setAddToLiquid
rescueBNB
convertBNBtoWBNB
transferBEP20Tokens
RescueBNBfromDistributor
TransferBEP20fromDistributor
AddToDistributorDeposit
AddToDistributorBalance
Withdrawal
Reinvest
setburnRatio
setstakepoolRatio
settaxRatio
ChangeMinHold
SetDistributionCriteria
ChangeImpoundTimelimit
SweepDivs
setStakePoolActive
setNFTPoolActive
changeContractGas
ChangeDistribGas
addPartnership
removePartnership
setEnablePartners
setPartnerFeeLimiter
setAirdropDisabled
setLaunchEnabled
```

Comments:

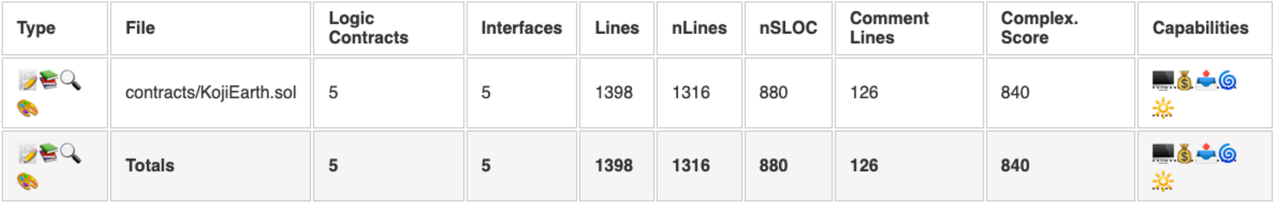
- manualBurn
    - Deployer can burn a certain number of tokens by transferring the amount to the dead address
- setInitialBlockLimit
    - Deployer can set initialBlocklimit without any limitations
- setBuyTxLimit
    - Deployer can set _maxTxAmountBuy without any limitations
- setBot
    - Deployer can set address as bot
- setIsDividendExempt
    - Deployer can exempt holder
- rescueBNB
    - This will allow owner to rescue BNB sent by mistake directly to the contract
- setburnRatio
    - Deployer can set burnRatio lower than taxRatio divided by 2
- setEnablePartners
    - Deployer can enable/disable partners

CallGraph



Source Units in Scope

v1.0



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	contracts/KojiEarth.sol	5	5	1398	1316	880	126	840	🖥💰📥🌀☀️
📝📚🔍🎨	**Totals**	**5**	**5**	**1398**	**1316**	**880**	**126**	**840**	🖥💰📥🌀☀️

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
nSLOC	normalized source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

AUDIT PASSED

Critical issues

- no critical issues found -

High issues

- no high issues found -

Medium issues

- no medium issues found -

Low issues

Issue	File	Type	Line	Description
#1	Main	Contract doesn’t import npm packages from source (like OpenZeppelin etc.)	-	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	Main	A floating pragma is set	45	The current pragma Solidity directive is „“^0.8.9””.
#3	Main	Missing Zero Address Validation (missing-zero-check)	135, 377,	Check that the address is not zero

#4	Main	State variable visibility is not set	257, 267, 268, 270, 272, 273, 274, 277, 288, 289, 290, 296, 298, 680, 681, 682, 684, 690, 703, 704, 706, 707, 709, 710, 711, 712, 714, 751, 753, 754, 755, 756	It is best practice to set the visibility of state variables explicitly
#5	Main	Missing Events Arithmetic	576, 362, 668, 320, 1264, 1052, 1097, 1047, 1379, 1057, 1198, 1203, 1208	Emit an event for critical parameter changes
#6	Main	Unchecked tokens transfer	634, 1157	Use `SafeERC20`, or ensure that the transfer/ transferFrom return value is checked
#7	Main	Tautology or contradiction	1099	Fix the incorrect comparison by changing the value type or the comparison

Informational issues

Issue	File	Type	Line	Description
#1	Main	State variables that could be declared constant (constable-states)	286, 681, 682, 690, 756, 722	Add the `constant` attributes to state variables that never change
#2	Main	Better variable description	831	Don’t use letters for variables, always use words to describe your passing variable
#3	Main	Require statement error message is missing	844	Add an error message to the require statement

#4	Main	Costly operations in a loop	593, 502, 468	currentIndex ++ Use a local variable to hold the loop computation result
#5	Main	Unused state variables	684, 756	Remove unused state variables

Commented Code exist

There are some instances of code being commented out in the following files that should be removed:

Line	Comment
83	// assert(a == b * c + a % b); // There is no case in which this doesn't hold

Recommendation

Remove the commented code, or address them properly.

Audit Comments

30. November 2021:

- Deployer can lock user funds
- For more information please read report

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
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	NOT 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	**NOT 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**

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY