



## Manual Smart Contract Audit



Low-Risk  
low-risk code

Medium-Risk  
medium-risk code

High-Risk  
high-risk code

### Contract Address

0x3b556DEe236E220DEdb8549a732580CdAE879df6 BSC

Disclaimer CodiState is not responsible for any financial losses. Nothing in this contract audit  
is financial advice, please do your own research.

# Disclaimer

CodiSecure is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

CodiSecure is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. CodiSecure does not endorse, recommend, support or suggest to invest in any project.

CodiSecure can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

## Tokenomics

BSC

## Source Code

- CodiSecure was comisioned by KlaToken to perform an audit based on the following smart contract:
  - <https://www.bscscan.com/token/0x3b556DEe236E220DEdb8549a732580CdAE879df6>
- BSC NETWORK

# About the Project and Company

KiaToken



Website: <https://kiahelps.com/>

## Manual Code Review

In this audit report we will highlight all these issues:

- Low-Risk

Deal with low-risk code issues

- Medium-Risk

Deal with medium-risk code issues

- High-Risk

Deal with high-risk code issues f

The detailed report continues on the next page...

# 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 they were discovered.

## Methodology

The auditing process follows a routine series of steps:

Code review that includes the following:

Review of the specifications, sources, and instructions provided to CodiSecure to make sure we understand the size, scope, and functionality of the smart contract.

Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.

Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to CodiSecure describe.

Testing and automated analysis that includes the following:

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.

Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

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.

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

```
contract KIATOKEN is PausableToken {  
  
    string public name;  
    string public symbol;  
    uint public decimals;  
    event Mint(address indexed from, address indexed to, uint256 value);  
    event Burn(address indexed burner, uint256 value);  
  
    constructor(string memory _name, string memory _symbol, uint256 _decimals,  
    uint256 _supply, address tokenOwner)  
  
public {      name = _name;  
    symbol = _symbol;  
    decimals = _decimals;  
    totalSupply = _supply * 10**_decimals;  
    balances[tokenOwner] = totalSupply;  
    owner = tokenOwner;  
    emit Transfer(address(0), tokenOwner, totalSupply);    }  
  
function burn(uint256 _value) public (_burn(msg.sender, _value));      }  
  
function _burn(address _who, uint256 _value) internal {  
    require(_value <= balances[_who]);  
    balances[_who] = balances[_who].sub(_value);  
    totalSupply = totalSupply.sub(_value);  
    emit Burn(_who, _value);  
emit Transfer(_who, address(0), _value);      }  
  
function mint(address account, uint256 amount) onlyOwner public {  
  
    totalSupply = totalSupply.add(amount);  
    balances[account] = balances[account].add(amount);  
    emit Mint(address(0), account, amount);  
    emit Transfer(address(0), account, amount);  
}  
}
```

## Tested Contract Files

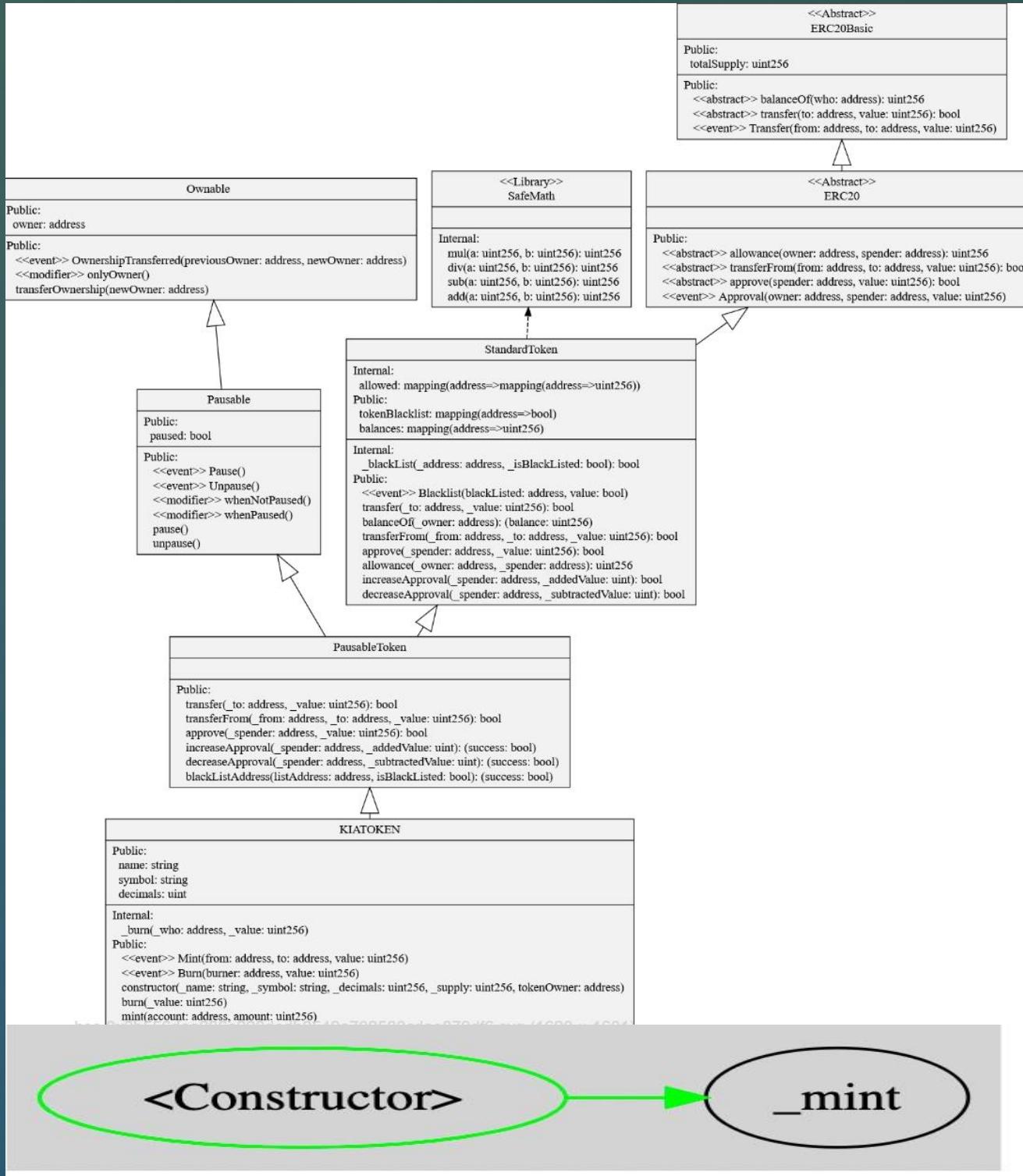
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
Contract/kiatoken.sol	723bcb577169ba64652075de5924383c

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

Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/ERC20.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.5.0/contracts/token/ERC20/ERC20.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.5.0/contracts/token/ERC20/ERC20.sol</a>

# Metrics



⌘ Public	⌘ Payable		
0	0		
External	Internal	Pure	View
0	1	0	0
Total	⌘ Public		
0 	0 		

## Metrics / Capabilities

 Exposed Functions	<input type="checkbox"/> Experimental Features	 StateVariables		 Has Destroyable Contracts
Solidity Versions observed 	 Can Receive Funds			 Uses Assembly
				

 Transfers  
ETH

  New/Create/Create2

<input type="checkbox"/> Low-Level Calls	 DelegateCall	<input type="checkbox"/> Uses Hash Functions	 ECRecover
--	--	--	---

## Exposed functions

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

# Contract Snapshot

Following contracts with the direct imports has been tested: kiatoken.sol

The audit team put forward the following assumption regarding the security and usage of the contract:

The contract is using the ERC-20 token standard

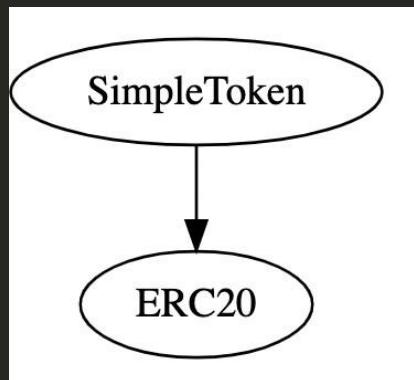
Owner/Deployer cannot mint new tokens

Owner/Deployer cannot burn or lock user funds

Owner/Deployer cannot pause the contract

The smart contract is coded according to the newest standards and in a secure way

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



# Manual and Automated Vulnerability Test

## CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

## HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

## MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract.

## LOW ISSUES

During the audit, Chainsulting's experts found **no Low issues** in the code of the smart contract.

## INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **no Informational issues** in the code of the smart contract.

# SWC Attacks

ID	Title		Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	☒
SWC-130	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	☒
SWC-129	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	☒
SWC-128	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	☒
SWC-127	Arbitrary Jump with Function TypeVariable	<a href="#">CWE-695: Use of Low-Level Functionality</a>	☒
SWC-125	Incorrect Inheritance Order	<a href="#">CWE-696: Incorrect Behavior Order</a>	☒
SWC-124	Write to Arbitrary Storage Location	<a href="#">CWE-123: Write-what-where Condition</a>	☒
SWC-123	<b>Requirement Violation</b>	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	☒

ID	Title		Test Result
SWC-113	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	☒
SWC-112	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	☒
SWC-111	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	☒
SWC-110	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	☒
SWC-109	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	☒
SWC-108	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	☒
SWC-107	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	☒
SWC-106	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	☒
SWC-105	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	☒
SWC-104	<b>Unchecked Call Return Value</b>	<a href="#">CWE-252: Unchecked Return Value</a>	☒

# Owner privileges

## ◎ Verify Claims

The contract is using the ERC-20 token standard

Status: tested and verified ◎

    Owner/Deployer cannot mint new tokens

Status: tested and verified ◎

    Owner/Deployer cannot burn or lock user funds

Status: tested and verified ◎

    Owner/Deployer cannot pause the contract

Status: tested and verified ◎

    The smart contract is coded according to the newest standards  
and in a secure way.

Status: tested and verified ◎

## Executive Summary

Two (2) independent SkillSCodified experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs

The overall code quality is good and not overloaded with unnecessary functions,  
these is greatly

benefiting the security of the contract. It correctly implemented widely used and  
reviewed contracts from OpenZeppelin.

The main goal of the audit was to verify the claims regarding the security of the smart  
contract and the claims inside the scope of work.

During the audit, no issues were found after the manual and automated security  
testing.

## Deployed Smart Contract

VERIFIED

<https://www.bscscan.com/address/0x3b556DEe236E220DEdb8549a732580CdAE879df6#code>

# About the Auditor

CodiSecure is a professional software development firm . that provides comprehensive distributed ledger technology (DLT) solutions. Some of their services include blockchain development, smart contract audits and consulting. CodiSecure conducts code audits on market-leading blockchains such as Hyperledger, Tezos, Ethereum, Binance Smart Chain, and Solana to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of 1Inch, POA Network, Unicrypt, Amun, Furucombo among numerous other top DeFi projects.

The blockchain security provider brings the highest security standards to crypto and blockchain platforms, helping to foster growth and transparency within the fast-growing ecosystem.

Check our website for further information:  
<https://www.skillscoidified.com>

