CodiSecure

AUDIT

Requirement Checker

□ Low-Risk

low-risk code

□ Medium-Risk

medium-risk code

□ High-Risk

high-risk code

Contract Address

LiaizonToken TEST 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. Coinsult 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 comissioned by LiaizonToken to perform an audit based on the following smart contract:

- https://testnet.bscscan.com/address/0x5615152f8e80934e8b532d45b2d4b3d5fe94b33c

- BSC TEST NETWORK

Contract Manual Code

```
Contract LiaizonToken  is Pausable, ERC20, BlackList {

    constructor() {
        totalSupply = 10000000;
        circulatingSupply = 0;
        name = "Liaizon Token";
        symbol = "Liaizon";
        decimals = 18;
        deprecated = false;
    }


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

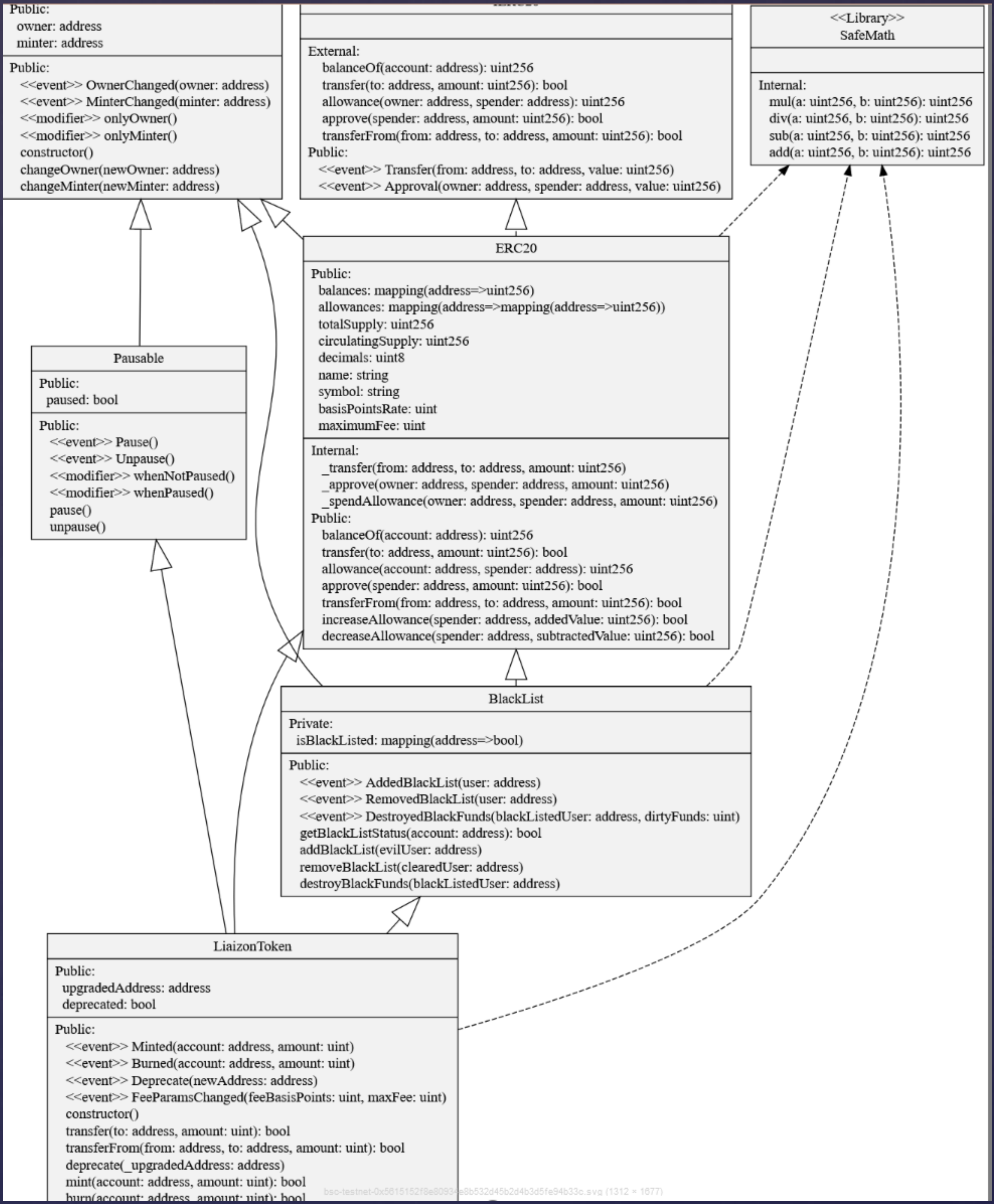
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5
Contract/LiaizonToken.sol	b2364a16da22d884d7abae82d483c423

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

Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.5.0/contracts/token/ERC20/ERC20.so

Metrics



```
Public:
  owner: address
  minter: address
Public:
   <<event>> OwnerChanged(owner: address)
   <<event>> MinterChanged(minter: address)
   <<modifier>> onlyOwner()
   <<modifier>> onlyMinter()
   constructor()
   changeOwner(newOwner: address)
   changeMinter(newMinter: address)
```

```
External:
    balanceOf(account: address): uint256
    transfer(to: address, amount: uint256): bool
    allowance(owner: address, spender: address): uint256
    approve(spender: address, amount: uint256): bool
    transferFrom(from: address, to: address, amount: uint256): bool
Public:
   <<event>> Transfer(from: address, to: address, value: uint256)
   <<event>> Approval(owner: address, spender: address, value: uint256)
```

```
<<Library>>
SafeMath

Internal:
   mul(a: uint256, b: uint256): uint256
   div(a: uint256, b: uint256): uint256
   sub(a: uint256, b: uint256): uint256
   add(a: uint256, b: uint256): uint256
```

```
ERC20
Public:
  balances: mapping(address=>uint256)
  allowances: mapping(address=>mapping(address=>uint256))
  totalSupply: uint256
  circulatingSupply: uint256
  decimals: uint8
  name: string
  symbol: string
  basisPointsRate: uint
  maximumFee: uint
Internal:
    _transfer(from: address, to: address, amount: uint256)
    _approve(owner: address, spender: address, amount: uint256)
    _spendAllowance(owner: address, spender: address, amount: uint256)
Public:
    balanceOf(account: address): uint256
    transfer(to: address, amount: uint256): bool
    allowance(account: address, spender: address): uint256
    approve(spender: address, amount: uint256): bool
    transferFrom(from: address, to: address, amount: uint256): bool
    increaseAllowance(spender: address, addedValue: uint256): bool
    decreaseAllowance(spender: address, subtractedValue: uint256): bool
```

```
Pausable
Public:
  paused: bool
Public:
   <<event>> Pause()
   <<event>> Unpause()
   <<modifier>> whenNotPaused()
   <<modifier>> whenPaused()
   pause()
   unpause()
```

```
BlackList
Private:
  isBlackListed: mapping(address=>bool)
Public:
   <<event>> AddedBlackList(user: address)
   <<event>> RemovedBlackList(user: address)
   <<event>> DestroyedBlackFunds(blackListedUser: address, dirtyFunds: uint)
   getBlackListStatus(account: address): bool
   addBlackList(evilUser: address)
   removeBlackList(clearedUser: address)
   destroyBlackFunds(blackListedUser: address)
```

```
LiaizonToken
Public:
  upgradedAddress: address
  deprecated: bool
Public:
   <<event>> Minted(account: address, amount: uint)
   <<event>> Burned(account: address, amount: uint)
   <<event>> Deprecate(newAddress: address)
   <<event>> FeeParamsChanged(feeBasisPoints: uint, maxFee: uint)
   constructor()
   transfer(to: address, amount: uint): bool
   transferFrom(from: address, to: address, amount: uint): bool
   deprecate(_upgradedAddress: address)
   mint(account: address, amount: uint): bool
   burn(account: address, amount: uint): bool
```

bsc-testnet-0x5615152f8e80934e8b532d45b2d4b3d5fe94b33c.svg (1312 × 1677)

⚜ Public	⚜ Payable
0	0

External	Internal	Pure	View
0	1	0	0

Total	⚜ Public
0 📥	0 ⚡

Metrics / Capabilities

👥 *Exposed Functions*	□ Experimental Features	🌐*StateVariables*	💣	💣 Has Destroyable Contracts

Solidity Versions observed 🏷	💰 Can Receive Funds		🖥 Uses Assembly
	🖥		

📤 Transfers ETH	🌀 🌀 New/Create/Create2

□ Low-Level Calls	👥 DelegateCall	□ Uses Hash Functions	🔖 ECRecover

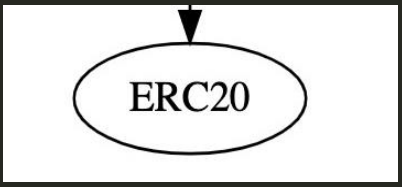
Exposed Functions

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 bacl fees
Owner/Deployer cannot pause the contract anti bot /landing/borw
The smart contract is coded according to the newest standards and in a secure way



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)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function TypeVariable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓
SWC-123	**Requirement Violation**	**CWE-573: Improper Following of Specification by Caller**	✓

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

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 COdisecure 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 TEST BSC Smart Contract

VERIFIED

https://testnet.bscscan.com/address/0x5615152f8e80934e8b532d45b2d4b3d5fe94b33c#code