VIDMA

Kingaru

SMART CONTRACT AUDIT

Project: Kingaru
Date: June 20th, 2022

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

During the audit, we analyzed the project's on-chain bridge logic. The Kingaru team has developed a cross-chain bridge that will allow users to transfer their tokens across the Binance Smart Chain, Kingaru blockchains, or any other supported chains. In order to claim their tokens on one of the desired blockchains, users will need to have a currency that is native to the chain of which they are claiming tokens.

For verification of cross-chain transfers involves off-chain logic run by the team and EIP712 signatures.

Please note, that we have not reviewed the off-chain logic related to the bridge.

During the audit process, the Vidma team found several issues with medium and informational severity levels. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	0 issues	0 issues	0 issues
High	0 issues	0 issues	0 issues
Medium	1 issue	1 issue	0 issues
Low	0 issues	0 issues	0 issues
Informational	1 issue	1 issue	0 issues
Total	**2 issues**	**2 issues**	**0 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are fully operational and secure. Under the given circumstances, we set the following risk level:

Our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Based on the **overall result of the audit**, the Vidma audit team grants the following score:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and the coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

Kingaru

A highly scalable, fast, secure blockchain for E-Commerce, NFT, Metaverse, and Retail.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from June 1st, 2022 to June 20th, 2022. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contracts:

- Bridge;
- KingaruBEP20;

The source code was taken from the following **source**:
https://bitbucket.org/applicature/jifu.contracts

Initial commit submitted for the audit:
b600b8e0adf178e9b1621ab8c598829b66e4e2f2

Last commit reviewed by the auditing team:
3ea94faf7cd00d8edba2c38ea5dfb6320ce6af1d

As a reference to the contracts logic, business concept, and the expected behavior of the codebase, the Kingaru team has provided the following documentation:
https://drive.google.com/drive/folders/1oROWAGh8UtNVqwBWJnrnq1bHh-4RsPhg?usp=sharing

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contract.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues. (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by Kingaru or not. The issues with “Not Valid” status are left on the list of findings but are not eligible for the score points deduction

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

Circulation supply of kingaru can exceed total supply

Medium | Resolved

Kingaru tokens for BSC will be minted with an initial supply of 1 billion tokens. This is the overall amount of tokens on the BSC and Kingaru chain. Part of Kingaru tokens should always be on the Kingaru chain for system contracts. So, the available amount of tokens on both chains can exceed the total Kingaru supply.

Recommendation:

Change KingaruBEP20.sol and instead of hardcoded initial supply use the number of tokens to be minted as a constructor parameter. If the Kingaru team has a mechanism to avoid this issue please share additional details.

Typos at Constants.sol and Bridge.sol

Informational | Resolved

In contract Constants.sol:
KYC_CONTAINER_TYPEHASE → *KYC_CONTAINER_TYPEHASH* (line 71).

In contract Bridge.sol:
CONTAINER_TYPEHASE → *CONTAINER_TYPEHASH* (line 32);
CONTAINER_KYC_TYPEHASE → *CONTAINER_KYC_TYPEHASH* (line 38);
CONTAINER_LIQUIDITY_TYPEHASE → *CONTAINER_LIQUIDITY_TYPEHASH* (line 43);
CONTAINER_LIQUIDITY_TYPEHASE → *CONTAINER_LIQUIDITY_TYPEHASE* (line 240);
CONTAINER_TYPEHASE → *CONTAINER_TYPEHASH* (line 399);
CONTAINER_KYC_TYPEHASE → *CONTAINER_KYC_TYPEHASH* (line 497).

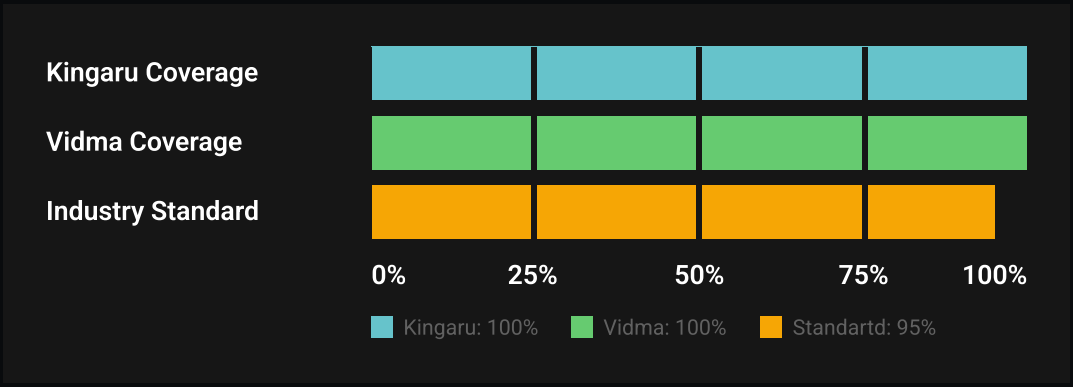
Recommendation:

Fix typos in variable names.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Truffle testing framework.

In this section, we provide both tests written by Kingaru and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the Kingaru's repository. We write totally separate tests with code coverage of a minimum of 95% to meet the industry standards.

Tests written by Kingaru

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
Bridge.sol	100.00	100.00	100.00	100.00
KingaruBEP20.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: Bridge
  Check correct initialized
    ✓ should fail if signers array is empty (45ms)
    ✓ should fail if fee is incorrect (50ms)
    ✓ Should correct initialized tokens types
    ✓ Should correct initialized tokens support address
  isTokenSupported
    ✓ Should return false if not support
    ✓ Should return true if support
  getSupportedTokens
    ✓ Should return correct list
    ✓ Should return empty after all delete (41ms)
    ✓ Should return correct list after add (48ms)
  addSigners
    ✓ Should fail if caller not owner
    ✓ Should fail if signer is zero address
    ✓ Should add correct signer
  getSignersAddress
    ✓ Should fail if caller not owner
```

```
  ✓ Should get correct list
removeSigners
  ✓ Should fail if caller not owner
  ✓ Should fail if signer for delete not found
  ✓ Should fail if signers array after delete is empty
  ✓ Should correct delete signer
setFeeDistributor
  ✓ Should correct set
  ✓ Should fail if caller not owner
setFeeRecipient
  ✓ Should be correct set
  ✓ Should fail if caller not owner
  ✓ Should fail if try setup zero address
proposeFee
  ✓ Should be correct send propose
  ✓ Should fail if fee more then max
  ✓ Should fail if try call from not owner
applyPropose
  ✓ Should fail if fee time has not
  ✓ Should fail if try call from not owner
  ✓ Should be apply propose and change coorect
removeSupportedToken
  ✓ Should fail if caller not owner
  ✓ Should fail if Token is not found
  ✓ Should correct delete and emit event (39ms)
addSupportedToken
  ✓ Should fail if caller not owner
  ✓ Should fail if fee is more or equal then 100%
  ✓ Should fail token type out of range
  ✓ Should correct emit event on add and set coorect field
  ✓ Should correct emit and change field if add before
    (fee should not change)
addLiquidity
  ✓ Should fail if expected two amount on input
  ✓ Should fail if amount is not bigger than zero
  ✓ Should fail if token is not supported
  ✓ Should correct transfer amount if native
  ✓ Should correct transfer amount if token
  ✓ Should correct change liquidity field (63ms)
  ✓ Should correct emit event
withdrawLiquidity
  ✓ Should fail if amount is zero (42ms)
```

```
    ✓ Should fail if amount is more then liquidity amount
    ✓ Should correct withdraw native and emit event (49ms)
    ✓ Should fail if time out (50ms)
    ✓ Should correct withdraw tokens and emit event (72ms)
    ✓ Should fail if nonce used before
    ✓ Should fail if invalid signer (39ms)
  deposit
    Should revert if
      ✓ token is not supported
      ✓ chain is not supported
      ✓ amount is zero when token is not native
      ✓ if eth chain and address is zero
      ✓ if not eth chain and data is empty
      ✓ two amount is set
      ✓ amount is zero when token is native
      ✓ native not enougth
    Should correct deposit
      ✓ native balance sc is empty
      ✓ erc20 sc is empty (46ms)
      native
        ✓ Should correct emit event
        ✓ Should correct transfer native to sc
        ✓ Should correct transfer native from user
      erc20 type transfer only
        ✓ Should correct emit event
        ✓ Should correct transfer erc20 to sc (46ms)
        ✓ Should correct transfer native from user
      erc20 type mint/burn v1
        ✓ Should correct emit event
        ✓ Should correct burn erc20 (46ms)
        ✓ Should correct transfer native from user
      erc20 type mint/burn v2
        ✓ Should correct emit event
        ✓ Should correct transfer erc20 to sc (43ms)
        ✓ Should correct transfer native from user
    Should correct deposite if kyc needed and have fee
      native
        ✓ invalid signer when kyc is nedeed (42ms)
        ✓ Should correct emit event
        ✓ Should correct transfer native to sc
        ✓ Should correct transfer native from user
      erc20 with kec and fee
```

```
    ✓ invalid signer when kyc is nedeed
    ✓ Should correct emit event
    ✓ Should call fee distributor
    ✓ Should correct transfer erc20 to sc
Withdraw
  Should correct withdraw
    native
      ✓ Should correct emit event
      ✓ Should correct transfer native from sc
      ✓ Should correct transfer native to user
    erc20 type withdraw transfer only
      ✓ Should correct emit event
      ✓ Should correct transfer erc20 to sc (47ms)
      ✓ Should correct transfer native from user
    erc20 type mint/burn v1/v2
      ✓ Should correct emit event
      ✓ Should correct burn erc20 (47ms)
      ✓ Should correct transfer native from user
  Should revert if
    ✓ nonce used before
    ✓ invalid signer
    ✓ erc20 amount not enoufg if erc20 type
    ✓ native not enougth
    ✓ token is not supported
    ✓ Arrays have different lengths
Withdraw with multi sign
  Should correct withdraw
    native
      ✓ Should correct emit event
      ✓ Should correct transfer native from sc
      ✓ Should correct transfer native to user
  Should revert if
    ✓ nonce used before
    ✓ invalid signer
    ✓ Arrays have different lengths
WithdrawFor
  Should correct withdraw
    native
      ✓ Should correct emit event
      ✓ Should correct transfer native from sc
      ✓ Should correct transfer native to user
    erc20 type withdraw transfer only
```

```
        ✓ Should correct emit event
        ✓ Should correct transfer erc20 to sc (46ms)
        ✓ Should correct transfer native from user
      erc20 type mint/burn v1/v2
        ✓ Should correct emit event
        ✓ Should correct burn erc20 (47ms)
        ✓ Should correct transfer native from user
    Should revert if
      ✓ nonce used before
      ✓ invalid signer
      ✓ erc20 amount not enoufg if erc20 type
      ✓ native not enougth
      ✓ token is not supported
      ✓ Arrays have different lengths
  withdrawForWithCostRecovery
    Should correct withdraw
      native
        ✓ Should correct emit event
        ✓ Should correct transfer native from sc
        ✓ Should correct transfer native to user
    Should revert if
      ✓ nonce used before
      ✓ invalid signer
      ✓ native not enougth
      ✓ native not enougth for fee (63ms)
      ✓ Arrays have different lengths
  WithdrawFor with multi sign
    Should correct withdraw
      native
        ✓ Should correct emit event
        ✓ Should correct transfer native from sc
        ✓ Should correct transfer native to user
    Should revert if
      ✓ nonce used before
      ✓ invalid signer (97ms)
      ✓ Arrays have different lengths

Contract: KingaruBEP20
  Deployment
    return state variables
      ✓ name
      ✓ symbol
```

```
      ✓ decimals
      ✓ getOwner
      ✓ totalSupply
  mint
    ✓ should fail if caller is not bridge
    ✓ should fail if total supply more than 1_000_000_000
    ✓ should mint tokens successfully (65ms)
  burn
    ✓ should fail if caller is not bridge
    ✓ should burn successfully (50ms)
  setBridgeAddress
    ✓ should fail if caller is not owner
    ✓ should fail if address of bridge is not contract
    ✓ should set address of bridge successfully (64ms)

144 passing (8s)
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
Bridge.sol	100.00	100.00	100.00	100.00
KingaruBEP20.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: Bridge
  Bridge Test Cases
    Bridge Deployment Test Cases
      ✓ shouldn't deploy with withdrawer setted to the zero
        address (82ms)
      ✓ shouldn't deploy with empty signers list (84ms)
      ✓ shouldn't deploy with signers setted to the zero
        address (69ms)
      ✓ shouldn't deploy with fee greater than max fee (71ms)
      ✓ should deploy with correct time of fee setting cooldown
      ✓ should deploy with correct max percentage
      ✓ should deploy with correct address for native currency
      ✓ should deploy with correct initial signers (71ms)
      ✓ should deploy with correct initial supported tokens (49ms)
      ✓ should deploy with correct initial supported tokens
        info (180ms)
  Bridge Owner Test Cases
    Supported Chains Test Cases
```

```
    ✓ shouldn't allow to set supported chain by not the current
      owner (77ms)
    ✓ should set supported chain for token correctly (101ms)
    ✓ should remove supported chain for token correctly (74ms)
  ETH Chains Test Cases
    ✓ shouldn't allow to set chain as ETH by not the current
      owner (56ms)
    ✓ should allow to set chain as ETH correctly (61ms)
    ✓ should allow to set chain as non ETH correctly (66ms)
  Signers Test Cases
    Signers Set Test Cases
      ✓ shouldn't allow to add signer by not the current
        owner (57ms)
      ✓ shouldn't allow to add signer with empty array of
        signers (62ms)
      ✓ shouldn't allow to add signer with zero address
      ✓ should add signer correctly (73ms)
      ✓ shouldn't allow to remove signer by not the current
        owner
      ✓ shouldn't allow to remove not existing signer
      ✓ shouldn't allow to remove all signers
      ✓ should remove signers correctly (53ms)
      ✓ shouldn't return list of signers if called by not
        the current owner (47ms)
      ✓ should return list of signers correctly (39ms)
    KYC Signer Test Cases
      ✓ shouldn't allow to set KYC signer by not the current
        owner (42ms)
      ✓ should set KYC signer correctly (70ms)
  Fee Recipient Test Cases
    ✓ shouldn't allow to set fee recipient by not the current
      owner (72ms)
    ✓ shouldn't allow to set fee recipient to the zero address
    ✓ should set fee recipient correctly (53ms)
  Fee Distributor Test Cases
    ✓ shouldn't allow to set fee distributor by not the current
      owner (65ms)
    ✓ should set fee distributor correctly (72ms)
  Supported Tokens Test Cases
    ✓ shouldn't allow to add new supported token by not the
      current owner
    ✓ should add new supported token correctly (55ms)
```

```
    ✓ should update supported token info if adding existing
      token correctly
    ✓ shouldn't allow to remove supported token by not the
      current owner (55ms)
    ✓ shouldn't allow to remove supported token if token not
      in list
    ✓ should remove supported token correctly (154ms)
  Fee Proposal/Apply Test Cases
    ✓ shouldn't allow to propose new fee for token by not the
      current owner
    ✓ shouldn't allow to propose new fee if fee percentage is
      greater than max fee
    ✓ should propose new fee correctly (55ms)
    ✓ shouldn't allow to apply fee propose for new fee if
      fee by not the current owner
    ✓ shouldn't allow to apply fee propose before cooldown
      period end
    ✓ should apply fee propose correctly (53ms)
  Withdrawer Test Cases
    ✓ shouldn't allow to set withdrawer by not the current owner
    ✓ shouldn't allow to set withdrawer to the zero address
    ✓ should set withdrawer correctly
Bridge Liquidity Test Cases
  Add Liquidity Test Cases
    Token Liquidity Test Cases
      ✓ shouldn't allow to add liquidity for not supported token
      ✓ shouldn't allow to add liquidity if two amount are
        entered (47ms)
      ✓ shouldn't allow to add token liquidity with zero
        amount (42ms)
      ✓ should add token liquidity correctly (114ms)
    Native Liquidity Test Cases
      ✓ shouldn't allow to add native liquidity with zero amount
      ✓ should add native liquidity correctly (52ms)
  Withdraw Liquidity Test Cases
    General Withdraw Liquidity Test Cases
      ✓ shouldn't allow to withdraw liquidity with expired
        deadline
      ✓ shouldn't allow to withdraw liquidity with different
        signature parameter array length (62ms)
      ✓ shouldn't allow to withdraw liquidity if signers length
        differs from signer parameter length (38ms)
```

```
      ✓ shouldn't allow to withdraw liquidity with incorrect
        signature (94ms)
    Withdraw Token Liquidity Test Cases
      ✓ shouldn't allow to withdraw token liquidity if withdraw
        amount is greater than available amount (97ms)
      ✓ shouldn't allow to withdraw token liquidity if withdraw
        amount is greater than max available amount (103ms)
      ✓ shouldn't allow to withdraw token liquidity if withdraw
        amount is 0 (102ms)
      ✓ should withdraw token liquidity correctly (267ms)
      ✓ shouldn't withdraw token liquidity with invalid
        nonce (61ms)
    Withdraw Native Liquidity Test Cases
      ✓ shouldn't allow to withdraw native liquidity if
        withdraw amount is greater than available amount (67ms)
      ✓ shouldn't allow to withdraw native liquidity if withdraw
        amount is greater than max available amount (80ms)
      ✓ shouldn't allow to withdraw native liquidity if withdraw
        amount is 0 (59ms)
      ✓ should withdraw native liquidity correctly (109ms)
  Bridge Deposit Test Cases
    General Deposit Test Cases
      ✓ shouldn't deposit token to not supported chain
      ✓ shouldn't deposit to ethereum based chain if recipient
        is the zero address
      ✓ shouldn't deposit to not ethereum based chain with
        empty data
      ✓ shouldn't deposit with incorrect KYC signature (140ms)
      ✓ shouldn't deposit unsupported token (159ms)
      ✓ shouldn't deposit if two amount was entered
      ✓ shouldn't deposit if amount is equal to 0
      ✓ should deposit ERC20 token types correctly (214ms)
      ✓ should deposit ERC20_MINT_BURN token types correctly (208ms)
      ✓ should deposit ERC20_MINT_BURN_V2 token types
        correctly (111ms)
      ✓ should deposit native currency correctly (177ms)
    Deposit Fee Test Cases
      ✓ should deposit native currency whith fee correctly (145ms)
      ✓ should deposit ERC20 token types with fee correctly (332ms)
      ✓ should deposit native currency whith fee and fee
        distributor setted correctly (270ms)
```

```
      ✓ should deposit token with fee and fee distributor setted
        correctly (441ms)
    Bridge Withdraw Test Cases
      General/User Withdraw Test Cases
        ✓ shouldn't withdraw unsupported token (78ms)
        ✓ shouldn't allow to withdraw with different signature
          parameter array length (118ms)
        ✓ shouldn't allow to withdraw if signers length differs from
          signer parameter length (94ms)
        ✓ shouldn't allow to withdraw with incorrect signature (66ms)
        ✓ should withdraw ERC20 token types correctly (117ms)
        ✓ should withdraw ERC20_MINT_BURN token types correctly (143ms)
        ✓ should withdraw ERC20_MINT_BURN_V2 token types
          correctly (202ms)
        ✓ should withdraw native currency correctly (102ms)
        ✓ shouldn't withdraw with used nonces (86ms)
      Bridge Withdraw For Test Cases
        ✓ should withdraw for user correctly (225ms)
      Bridge Withdraw For With Cost Recovery Test Cases
        ✓ shouldn't allow to withdraw if sender is not withdrawer
        ✓ shouldn't allow to withdraw if amount is less than
          tx fee (163ms)
        ✓ should withdraw with cost recovery correctly (111ms)
      Bridge View Test Cases
        ✓ should return correct list of supported tokens
        ✓ should return correct data for token supportment (64ms)

  Contract: KingaruBEP20
    Kingaru Test Cases
      Kingaru Deployment Test Cases
        ✓ shouldn't deploy with empty signers list (55ms)
        ✓ should deploy with correct owner
        ✓ should deploy with correct decimals (52ms)
        ✓ should deploy with correct total supply
        ✓ should deploy with correct max supply
        ✓ should deploy with correct bridge address (45ms)
      General Kingaru Test Cases
        ✓ shouldn't mint tokens by not the bridge contract
        ✓ shouldn't mint tokens more than max cap
        ✓ should mint tokens correctly
        ✓ shouldn't burn tokens by not the bridge contract
        ✓ should burn tokens correctly (51ms)
```

```
    ✓ shouldn't set new bridge contract by not the current
      owner
    ✓ shouldn't set new bridge contract if new bridge address
      not the contract (50ms)
    ✓ should set new bridge contract correctly
  Kingaru BEP20 Test Cases
    Transfer Test Cases
      ✓ should transfer tokens correctly (55ms)
      ✓ shouldn't transfer with zero sender address
      ✓ shouldn't transfer with zero recipient address
      ✓ shouldn't transfer more tokens than sender balance
    Approve Test Cases
      ✓ should approve tokens correctly
      ✓ shouldn't approve with zero spender address
    Allowance Test Cases
      ✓ should return allowance tokens correctly
      ✓ should increase allowance correctly
      ✓ should decrease allowance correctly (49ms)
      ✓ shouldn't decrease allowance if subtracted amount
        more than current allowance

121 passing (13s)
```

We are delighted to have a chance to work with the Kingaru team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA