VIDMA

Kingaru

SMART CONTRACT AUDIT

Project: Kingaru
Date: July 6th, 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.

Overall, the contracts are well-written and structured. The code style follows the best practices. During the auditing process, Vidma security team found a couple of issues with “informational” and “low” severity levels. Most of them were fixed by Kingaru team. Vidma auditors confirms that the contracts are risk-free and operational.

During the audit process, the Vidma team found several issues. 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	0 issues	0 issues	0 issues
Low	5 issues	5 issues	0 issues
Informational	6 issues	6 issues	0 issues
Total	**11 issues**	**11 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 9, 2022 to July 4, 2022. The outcome is disclosed in this document.

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

- KRUAPRStake;
- KRUBonusDistributor;
- KRUCoinExchangeToNative;
- KRUDiscountExchange;
- KRUExchangeForwarder;
- KRUForwarder;
- KRUReferralTree;
- KRUTokenPlan;
- WKRU;
- ManagementUpgradeable;
- Management;
- ManagedUpgradeable;
- Managed.

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:
1577a99aaebbe94b833251625c3c6444aeff4b21

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

Lack of validation in WKRU

Low | Resolved

In function *transferFrom()* in WKRU contract:

- missed checking for zero address for *sender_* and *recipient_*;
- missed checking for zero value for *amount_* param.

Recommendation:

Consider adding appropriate require statements.

State variable can be declared constant

Low | Resolved

There is a state variable in the WKRU contract such as name, symbol, and decimals which is never changed. Constant state variables should be declared constant to save gas.

Recommendation:

Consider adding the constant attributes to these variables.

Lack of zero address check

Low | Resolved

In function *forward()* in KRUForwarder contract missed checking for zero address for *receiver_ address*.

Recommendation:

Consider adding require to check if the param is the zero address.

Unhandled case when the recipient_ has no valid purchases

Low | Resolved

In contract KRUDiscountExchange.sol in function *getBalanceInfo()* there is no check if the recipient has a valid purchase at the index *indexOfPurchase_* which can lead to an unexpected error.

Recommendation:

Consider adding a check if the recipient has any purchase with the index *indexOfPurchase_* if no return 0 in this case.

Useless require statement

Low | Resolved

In function *setCommissionForwardTarget()* contract KRUFarwarder there is require statement where parameter *destination_* with enum type is checked if the diapason of possible options is correct.

```
require(uint8(destination_) < 2, "Wrong destination index")
```

As *destination_* param is an enum type Destination this already restricts a variable to have one of only a few predefined values. In this specific case, it is 1 and 2. So it is possible to avoid useless require to keep the clear code and decrease the gas usage for the function call.

Recommendation:

Consider removing the useless require statement.

Some kind of typos in the code source

Informational | Resolved

There are some typos in function and variable names.

Typos in contract KRUDiscountExchange.sol:

```
Function _recive => _receive()
```

```
constant  _CONTAINER_TYPEHASE  => _CONTAINER_TYPEHASH
```

Typos in contract KRUExchangeForwarder.sol:

```
constant  _CONTAINER_TYPEHASE  => _CONTAINER_TYPEHASH
```

Recommendation:

Consider fixing typos.

Comparison to boolean constants

Informational | Resolved

There is a comparison to boolean constants in the functions *removeDepositInfo()* and *forward()* contract KRUCoinExchangeToNative. Boolean constants can be used directly and do not need to be compared to *true* or *false*.

Recommendation:

Consider removing the equality to the boolean constant.

Inappropriate error message code

Informational | Resolved

In contract KRUDiscountExchange.sol in function *issuePurchase()* there is a check whether *dateOfPurchase_* is greater than zero require.

```
dateOfPurchase_ > 0, ERROR_WRONG_AMOUNT
```

It will be more appropriate to use here error code:

```
ERROR_AMOUNT_IS_ZERO
```

to be more specific in the error definition and avoid any misunderstanding.

Recommendation:

Consider changing the error code.

Useless conversions to type

Informational | Resolved

In function *addDepositAmount()* contract KRUCoinExchangeToNative parameter *plan_* is passed as argument and in the function implementation is casting to type Plan:

```
Plan plan = Plan(plan_)
```

It is possible to avoid the useless line of code and pass plan_ parameter already with the type Plan.

Recommendation:

Consider changing parameter *plan_* type to Plan.

Useless variables initialization

Informational | Resolved

In contract KRUReferralTree.sol there is some variable with useless initialization:

- In the constructor, the variable *maxRewardLevel* is initialized to 5 however, it is set to the same value when a variable is declared;
- In the function *createTree() node.ownPurchase* and *node.totalPurchase* is initializing to 0 what is useless as type uint256 by default equals to 0.

Recommendation:

Consider removing useless initialization to decrease the gas cost for calling functions and keep the best practice.

Functions visibility optimization

Informational | Resolved

In the contract WKRU.sol functions *withdraw()*, *approve()* and *transfer()* can be marked as external as they are not called anywhere inside the contract.

The same solution is possible to use with the *deposit()* function. It is possible to separate logic of the deposit function in internal function *deposit()* and call this internal function in the *receive()* and *deposit()* function. And in this case *deposit()* will be external.

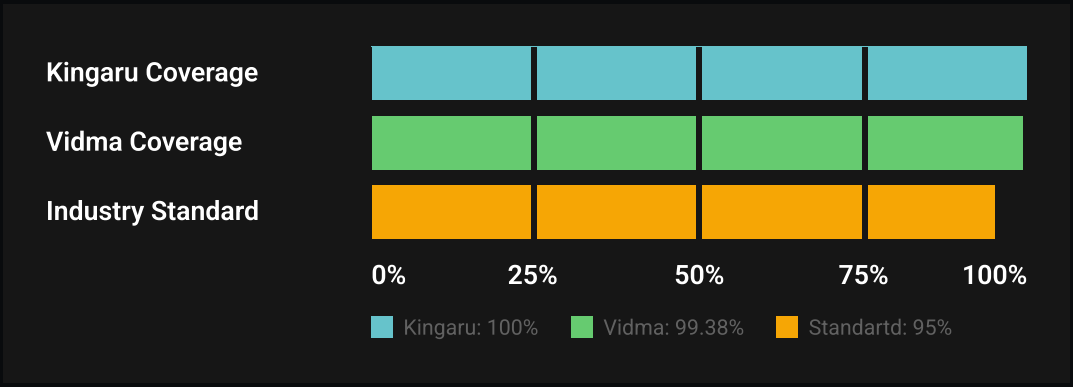
Recommendation:

Consider changing visibility from public to external.

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	97.66	100.00	100.00
KRUAPRStake.sol	100.00	100.00	100.00	100.00
KRUBonusDistributor.sol	100.00	100.00	100.00	100.00
KRUCoinExchangeToNative.sol	100.00	100.00	100.00	100.00
KRUDiscountExchange.sol	100.00	94.44	100.00	97.01
KRUExchangeForwarder.sol	100.00	93.75	100.00	100.00
KRUForwarder.sol	100.00	90.00	100.00	100.00
KRUReferralTree.sol	100.00	92.86	100.00	100.00
KRUTokenPlan.sol	100.00	100.00	100.00	100.00
WKRU.sol	100.00	100.00	100.00	100.00
contracts\management\	100.00	94.44	100.00	100.00
Managed.sol	100.00	87.5	100.00	100.00
ManagedUpgradeable.sol	100.00	90.00	100.00	100.00
Management.sol	100.00	90.00	100.00	100.00

File	% Stmts	% Branch	% Funcs	% Lines
ManagementUpgradeable.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**97.2**	**100.00**	**100.00**

Test Results

```
Contract: KRUAPRStake
Your project has Truffle migrations, which have to be turned into
a fixture to run your tests with Hardhat
  Permissions
    ✓ CAN_STAKE_FOR_APR_STAKE (95ms)
    ✓ Grant permissions CAN_STAKE_FOR_APR_STAKE (73ms)
    ✓ Grant permissions CAN_STAKE_FOR_APR_STAKE (83ms)
  should be able to:
    ✓ receive JIFU coin
    ✓ should revert if sender not the owner (47ms)
    ✓ withdraw JIFU coin (53ms)
  Functions
    stake
    ✓ first purchase, all sent funds must be on stake balance
    ✓ second purchase, all sent funds must be on stake balance
    ✓ funds must be on apr contract address
    ✓ Can stake/unstake 0
    ✓ member balance after stake
    unstake
    ✓ must unstake amount
    ✓ funds must unstake from apr contract address
    ✓ can't unstake more than the stake balance
    ✓  can't unstake zero amount
  Distrubited more then deposite eth
    ✓ receive JIFU coin (116ms)
  should be correct calculate rewards
    ✓ after year (77ms)
    ✓ after update percentage more time (117ms)
    ✓ two user with restkae (216ms)
    ✓ check every month (408ms)
```

```
Contract: KRUBonusDistributor
  Function
    setBonusAmount
      ✓ Should be fail if not owner
      ✓ Check correct setup
    distributionCoins
      ✓ Check Permissions: CAN_DISTRIBUT_BONUS_KRU_DISTRIBUTOR
      ✓ Check if not enouth balance
      Should correct distribution
        ✓ Should be send correct amount
        ✓ Should be fail if set again in one address
        ✓ Should be emit correct event

Contract: KRUForwarder
  ✓ should revert if forward amount is zero
  ✓ should revert if not enough coin in reward pool
  Permissions CAN_FORWARD_FORWARDER
    ✓ should revert if dont have permission
    ✓ should revert if receiver is a zero address
    ✓ should pass if permission was granted
  should be able to:
    ✓ receive JIFU coin
    ✓ should revert if sender not the owner
    ✓ withdraw JIFU coin
  Set destination
    ✓ COMMISSION_BALANCE should be as default destination
    ✓ should set destination to APR Stake (43ms)
    ✓ Should revert if not correct index
  Getters
    ✓ destinations

Contract: KRUCoinExchangeToNative
  Functions
    getUserPurchasesInfo
      ✓ Should return user purchase info
      ✓ Should return user purchase info after removing
        purchase (107ms)
    addDepositAmount
      ✓ Should fail if sender has no permission (38ms)
      ✓ Should fail if token does not exist (47ms)
      ✓ Should add deposit amount of tokens correctly
    removeDepositInfo
```

```
      ✓ Should fail if sender has no permission
      ✓ Should fail if token does not exist
      ✓ Should fail if index was out of range (45ms)
      ✓ Should fail if purchased tokens were claimed (80ms)
      ✓ Should remove deposit information correctly (60ms)
    forward
      ✓ Should fail if sender has no tokens
      ✓ Should fail if token does not exist
      ✓ Should fail if the forward is not start (112ms)
      ✓ Should forward correctly (48ms)
      ✓ Should forward correctly but balance dont change (109ms)
    setSupportedTokens/setForwardTimeOfPlans
      ✓ Should fail if sender has no permission to
        setSupportedTokens
      ✓ Should fail if sender has no permission to
        setForwardTimeOfPlans
      ✓ Should fail if set different length setSupportedTokens
      ✓ Should fail if set different length setForwardTimeOfPlans
      ✓ Should set forward time of plans correctly
      ✓  Should set supported tokens correctly (38ms)

Contract: KRUReferralTree
  Referral tree
    Check permissions
      ✓ CAN_CREATE_TREE_REFERRAL_TREE (131ms)
      ✓ CAN_UPDATE_CALCULATE_REWARDS_REFERRAL_TREE (163ms)
    Check create tree
      ✓ ERROR_TREE_EXISTS (44ms)
      ✓ ERROR_TREE_DOESNT_EXIST when parent not exist (39ms)
      ✓ ERROR_NOT_DIFFERENT_MEMBERS when parent and member
        are the same
    Check reward for level
      ✓ Should transfer correct reward for level from tree (771ms)
    Modifier
      onlyForExistingTree
        ✓ Should fail with error : ERROR_TREE_DOESNT_EXIST
    Function
      getParent
        ✓ Should return correct parent address for exist tree
        ✓ Should fail for not exist tree
      getTotalPurchase
        ✓ Should return correct total purchase for exist tree (46ms)
```

```
      ✓ Should fail for not exist tree
    getMemberPurchase
      ✓ Should return correct own purchase for exist tree (58ms)
      ✓ Should fail for not exist tree
    getNetworkPurchase
      ✓ Should return correct network purchase for exist
        tree (66ms)
      ✓ Should fail for not exist tree
    setMaxRewardLevel
      ✓ Should call from only owner
      ✓ Should correct effect on rewards (76ms)
    setPercentagePerLevel
      ✓ Should call from only owner
      ✓ Should correct effect on rewards (83ms)
    getDirectChildrenCount
      ✓ Should return correct count of direct child for exist
        tree (57ms)
      ✓ Should fail for not exist tree
    getTree
      ✓ Should return correct tree for exist tree (74ms)
      ✓ Should fail for not exist tree
  Getters
    ✓ referralNodes
    ✓ maxRewardLevel
    ✓ percentagePerLevel

Contract: KRUCoinExchangeToNative
  Functions
    mint
      ✓ Should fail if sender has no permisson
      ✓ Should fail if amount of tokens - zero
      ✓ Should mint tokens successfully
    burn
      ✓ Should fail if sender has no permisson
      ✓ Should fail if the given amount of tokens is more than
        balance of user (49ms)
      ✓ Should burn tokens successfully
    transfer
      ✓ Should fail if transfer tokens to another address
      ✓ Should fail if transferFrom tokens to another address
      ✓ Should fail if transfer tokens to the zero address
```

```
Contract: KRUDiscountExcange
  initialize
    ✓ should fail deploy KRUDiscountExcange if management contract
      is zero (46ms)
    ✓ should correct add init coins for issuance
  coinsForIssuance
    ✓ should add to variable sum which will transfer to contract
    ✓ should remove form variable sum which will withrdaw
      from contract (47ms)
  addSigners
    ✓ should fail if member call addSigners without permission
    ✓ should fail if length of array is zero
    ✓ should add signers successfully
  removeSigners
    ✓ should fail if member call removeSigners without permission
    ✓ should fail if signer is not exist
    ✓ should fail if array length of signers is zero (50ms)
  createVestingTypes
    ✓ should fail if member call createVestingTypes without
      permission
    ✓ should fail if member call createVestingTypes without
      permission
    ✓ should create new vesting type successfully
  disableVestingType
    ✓ should fail if member call disableVestingType without
      permission
    ✓ should fail if index of vesting type does not exist or zero
    ✓ should remove vesting type successfully
  issuePurchase
    ✓ should fail if try issuence more coins than have on balance
    ✓ should fail if member call issuePurchase without permission
    ✓ should fail if index of vesting type is out
    ✓ should fail if different array length (53ms)
    ✓ should fail if signer is invalid or signers passed in
      different positions (60ms)
    ✓ should fail if date of purchase is zero (42ms)
    ✓ should transfer native coin to recipient successfully (108ms)
    ✓ should set vesting to user successfully (125ms)
  claim/claimFor
    ✓ should fail if index of purchase is out
    ✓ should fail if amount of rewards is zero
    ✓ should claim / claim for recipient successfully (152ms)
```

```
  vesting test cases
    ✓ user has been purchase native coin twice with the different
      vesting type (300ms)

Contract: KRUExchangeForwarder
  initialize
    ✓ should fail if destination address is zero (43ms)
    ✓ should fail deploy KRUExchangeForwarder if management
      contract is zero (58ms)
    ✓ should deploy KRUShopsManager successfully (40ms)
  setDestinationAddress
    ✓ should fail if member call setDestinationAddress without
      permission
    ✓ should fail if destination address is zero
    ✓ should set destination address successfully
  setAvailableTokens
    ✓ should fail if member call setAvailableTokens without
      permission
    ✓ should set available tokens successfully
  forward
    forwardNative
      ✓ should fail if zero address is not set in available
        tokens (43ms)
      ✓ should fail if deadline has already passed
      ✓ should fail if sender is not message signer
      ✓ should fail if message signer has already forwarded coins
        by that signature (41ms)
      ✓ should fail if amount of native coin is zero
      ✓ should forward native coin correctly (63ms)
    forwardToken
      ✓ should fail if passed token is not in list of available
        tokens (43ms)
      ✓ should fail if deadline has already passed
      ✓ should fail if sender is not message signer
      ✓ should fail if message signer has already forwarded tokens
        by that signature (90ms)
      ✓ should fail if amount of tokens is zero
      ✓ should forward tokens correctly (121ms)

Contract: Managed
  Deploy
    ✓ Should initialize field correct
```

```
    ✓ Should fail if management contract address will be zero
  Functiots
    setManagementContract
      ✓ Should fail if caller not owner
      ✓ Should fail if try setup zero_address
      ✓ Should correct setup Management
    hasPermission
      ✓ Should correct return
      ✓ Should fail if caller haven't permissions
    canCallOnly
      ✓ Should correct return
      ✓ Should fail if caller call from not registered contract

Contract: ManagedUpgradeable
  Deploy
    ✓ Should initialize field correct
    ✓ Should fail if try initialize secondly
    ✓ Should fail if try initialize zero address (84ms)
  Functions
    setManagementContract
      ✓ Should fail if caller not owner
      ✓ Should fail if try setup zero_address
      ✓ Should correct setup Management
    hasPermission
      ✓ Should fail if caller haven't permissions
      ✓ Should correct return
    canCallOnly
      ✓ Should correct return
      ✓ Should fail if caller call from not registered contract

Contract: Management
  Functions
    registerContract
      ✓ Should fail if users can owner
      ✓ Should correct register contract
    permissions
      ✓ Should return correct if permissions user have
    setLimitSetPermission
      ✓ Should fail if users can owner
      ✓ Should correct set limit permission
      ✓ Should correct set limit permission - false (49ms)
    setPermissions
```

```
      ✓ Should fail if users can owner
      ✓ Should correct set permissions
    setPermission
      ✓ Should fail if users can owner
      ✓ Should correct set permissions
      ✓ Should correct set permission from not owner but with limit
        set permissions true
      ✓ Should fail if try call from user with limit but with
        another permissions which allow
    setKycWhitelists
      ✓ Should fail if users can have
        MANAGEMENT_CAN_SET_KYC_WHITELISTED
      ✓ Should correct set kyc whitelisted
    isKYCPassed
      ✓ fail if time out of sign
      ✓ return true if addres have MANAGEMENT_WHITELISTED_KYC
      ✓ return true if user correct push sign
      ✓ fail if user correct push invalid sign (41ms)
      ✓ fail if time out of sign

Contract: ManagementUpgradeable
  Functions
    registerContract
      ✓ Should fail if users can owner
      ✓ Should correct register contract
    permissions
      ✓ Should return correct if permissions user have
    setLimitSetPermission
      ✓ Should fail if users can owner
      ✓ Should correct set limit permission
      ✓ Should correct set limit permission - false (53ms)
    setPermissions
      ✓ Should fail if users can owner
      ✓ Should correct set permissions
    setPermission
      ✓ Should fail if users can owner
      ✓ Should correct set permissions
      ✓ Should correct set permission from not owner but with limit
        set permissions true (70ms)
      ✓ Should fail if try call from user with limit but with
        another permissions which allow (259ms)
    setKycWhitelists
```

```
      ✓ Should fail if users can have
        MANAGEMENT_CAN_SET_KYC_WHITELISTED
      ✓ Should correct set kyc whitelisted
    isKYCPassed
      ✓ fail if time out of sign
      ✓ return true if addres have MANAGEMENT_WHITELISTED_KYC
      ✓ return true if user correct push sign
      ✓ fail if user correct push invalid sign (43ms)
      ✓ fail if time out of sign

Contract: KRUShopsPaymentProccesor
  Deployment
    return state variables
      ✓ name
      ✓ symbol
      ✓ decimals
      ✓ totalSupply
  Functions
    receive
      ✓ should make deposit by fallback receive (46ms)
      ✓ should fail if amount is zero
    deposit
      ✓ should deposit funds only with deposit method (40ms)
      ✓ should fail if amount is zero
    withdraw
      ✓ should withdraw funds correctly (38ms)
      ✓ should fail if balance amount is not enough
    transfer
      ✓ should transfer correctly
    approve
      ✓ should approve correctly
    transferFrom
      ✓ should transferFrom correctly (45ms)
      ✓ should fail if balance amount is not enough
      ✓ should fail if not enough allocation

215 passing (26s)
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	99.15	98.00	98.5	99.10
KRUAPRStake.sol	100.00	100.00	100.00	100.00
KRUBonusDistributor.sol	100.00	100.00	100.00	100.00
KRUCoinExchangeToNative.sol	100.00	100.00	100.00	100.00
KRUDiscountExchange.sol	97.06	97.22	92.86	97.01
KRUExchangeForwarder.sol	100.00	93.75	100.00	100.00
KRUForwarder.sol	100.00	90.00	100.00	100.00
KRUReferralTree.sol	100.00	92.86	100.00	100.00
KRUTokenPlan.sol	100.00	100.00	100.00	100.00
WKRU.sol	100.00	100.00	100.00	100.00
contracts\management\	100.00	97.22	100.00	100.00
Managed.sol	100.00	87.5	100.00	100.00
ManagedUpgradeable.sol	100.00	100.00	100.00	100.00
Management.sol	100.00	100.00	100.00	100.00

File	% Stmts	% Branch	% Funcs	% Lines
ManagementUpgradeable.sol	100.00	100.00	100.00	100.00
All Files	**99.38**	**98.40**	**98.45**	**99.25**

Test Results

```
Contract: KRUCoinExchangeToNative
  setSupportedTokens
    ✓ should revert with error access denied, caller is not
      admin (565ms)
    ✓ should revert with error access denied, caller is not
      admin (267ms)
    ✓ should set correctly (316ms)
    ✓ should catch event
  setForwardTimeOfPlans
    ✓ should revert with error access denied, caller is not
      admin (278ms)
    ✓ should revert with error diff arr length (276ms)
    ✓ should set correctly (248ms)
    ✓ should catch event
  addDepositAmount
    ✓ should revert with error access denied, caller is not
      admin (305ms)
    ✓ should revert with error token not supported (302ms)
    ✓ should add correctly (637ms)
    ✓ should catch event
  removeDepositInfo
    ✓ should revert with error access denied, caller is not
      admin (244ms)
    ✓ should revert with error token not supported (279ms)
    ✓ should revert with error index out (257ms)
    ✓ should remove correctly (407ms)
  forward
    ✓ should revert with error token not supported (274ms)
    ✓ should revert with error amount is zero (280ms)
    ✓ should revert with error not start (1441ms)
```

```
    ✓ should forward correctly (1214ms)

Contract: KRUExchangeForwarder
  initialize
    ✓ cannot initialize twice (197ms)
    ✓ should revert: destination address is zero (651ms)
    ✓ should initialize correctly (620ms)
  forward
    ✓ should revert with error access denied, token is not
      available (230ms)
    ✓ should revert with error invlid signer (275ms)
    ✓ should revert with error zero amount (289ms)
    ✓ should revert with error time out (402ms)
    ✓ should forward when token is not zero address (797ms)
    ✓ should catch event
    ✓ should forward when token is zero address (285ms)
    ✓ should catch event
    ✓ should revert with error access denided (355ms)
  setDestinationAddress
    ✓ should revert with error access denied, caller doesn't has
      permissions (477ms)
    ✓ should revert with error invalid address (220ms)
    ✓ should set destination correctly (376ms)
    ✓ should catch event
  setAvailableTokens
    ✓ should revert with error access denied, caller doesn't has
      permissions (260ms)
    ✓ should set token correctly (354ms)
    ✓ should catch event

Contract: KRUForwarder
  setCommissionForwardTarget
    ✓ should set correctly (363ms)
    ✓ should catch event
  forward
    ✓ should revert with error access denied, caller is not
      admin (225ms)
    ✓ should revert with error receiver can't be zero address (293ms)
    ✓ should revert with error can't forward 0 (407ms)
    ✓ should revert with error not enough funds to forward (284ms)
    ✓ should forward correctly if destinat is APR_STAKE (1196ms)
```

```
    ✓ should forward correctly if destinat is
      COMMISSION_BALANCE (786ms)
    ✓ should catch event

Contract: KRUTokenPlan
  mint
    ✓ should revert with error access denied, caller can't
      mint (233ms)
    ✓ should revert with error amount is zero (609ms)
    ✓ should mint correctly (491ms)
  burn
    ✓ should revert with error access denied, caller can't
      burn (535ms)
    ✓ should revert with error when user don't have tokens
      to burn (304ms)
    ✓ should burn correctly (899ms)
  transfer
    ✓ should revert with error access denied, receiver address is
      not correct (478ms)
    ✓ should revert with error access denied, sender address is
      not correct (731ms)
    ✓ shouldn't transfer

Contract: KRUDiscountExcange
  createVestingTypes
    ✓ should revert with error access denied, caller is not
      admin (748ms)
    ✓ should revert with error length is zero (273ms)
    ✓ should create new vesting types correctly (3097ms)
  disableVestingType
    ✓ should revert with error access denied, caller is not
      admin (1316ms)
    ✓ should revert with error index out (394ms)
    ✓ should set vesting as disable correctly (419ms)
  addSigners
    ✓ should revert with error access denied, caller is not
      admin (681ms)
    ✓ should revert with error length is zero (400ms)
    ✓ should add signers correctly (541ms)
  removeSigners
    ✓ should revert with error access denied, caller is not
      admin (283ms)
```

```
    ✓ should revert with error not found (342ms)
    ✓ should revert with error length is zero (440ms)
    ✓ should remove signers correctly (658ms)
  issuePurchase
    ✓ should revert with error access denied, caller doesn't has
      permissions (543ms)
    ✓ should revert with error index out (633ms)
    ✓ should revert with error amount is zero (965ms)
    ✓ should revert with error wrong amount (250ms)
    ✓ should revert with error not enough balance (270ms)
    ✓ should revert with error access denied, vesting is
      disabled (565ms)
    ✓ should revert with error diff arr length (543ms)
    ✓ should revert with error invalid signer (696ms)
    ✓ should work correctly when cliff duration == 0 and
      initialPercentage == 100 and make withdraw (1173ms)
    ✓ should catch events
    ✓ should work correctly in another case (1164ms)
    ✓ should catch event
    ✓ should revert with error invalid nonce (372ms)
  claim
    ✓ should revert with error index out, purchase does not
      exist (188ms)
    ✓ should revert with error amount is zero (273ms)
    ✓ should claim correctly (822ms)
    ✓ should catch event
  claimFor
    ✓ should revert, user don't have access for claim (319ms)
    ✓ should revert with error index out, purchase does not
      exist (264ms)
    ✓ should revert with error amount is zero (265ms)
    ✓ should claim correctly (1314ms)
    ✓ should catch event

Contract: KRUBonusDistributor
  setBonusAmount
    ✓ should revert with error access denied, caller is not
      admin (354ms)
    ✓ should set correctly (236ms)
    ✓ should catch event
  distributionCoins
```

```
    ✓ should revert with error access denied, caller is not
      admin (308ms)
    ✓ should revert with error not enough balance (253ms)
    ✓ should distribute correctly (814ms)
    ✓ should catch event
    ✓ should revert with error already distributed (260ms)

Contract: KRUAPRStake
  setPercentage
    ✓ should revert with error access denied, only owner can
      update percentage (273ms)
    ✓ should set percentage correctly (862ms)
    ✓ should catch event
  stake
    ✓ should revert with error amount is zero (252ms)
    ✓ should stake correctly (690ms)
    ✓ should stake few times correctly (1178ms)
    ✓ should catch event
  stakeFor
    ✓ should revert with error access denied, user can't
      stake (464ms)
    ✓ should revert with error amount is zero (225ms)
    ✓ should stake correctly (695ms)
    ✓ should stake few times correctly (1350ms)
    ✓ should catch event
  unstake
    ✓ should revert with error amount is zero (308ms)
    ✓ should revert with error more than max, user can't unstake
      more than stake balance (193ms)
    ✓ should unstake correctly (1128ms)
    ✓ should catch event
  stake, unstake, withdraw interest
    ✓ should work correctly for few users (12602ms)

Contract: KRUReferralTree
  createTree
    ✓ should revert with error access denied, caller is not
      admin (687ms)
    ✓ should revert with error not different members (264ms)
    ✓ should revert with error tree exists (586ms)
    ✓ should revert with error tree doesnt exist (334ms)
    ✓ should create tree and catch event (319ms)
```

```
  calculateUpdateRewards
    ✓ should revert with error access denied, caller is not
      admin (333ms)
    ✓ Should calculate rewards correctly foreach tree level (15742ms)
    ✓ should catch event
  setMaxRewardLevel
    ✓ should revert with error access denied, caller is not
      admin (271ms)
    ✓ should set reward level and catch event (287ms)
  setPercentagePerLevel
    ✓ should revert with error access denied, caller is not
      admin (261ms)
    ✓ should percentage level and catch event (365ms)
  getTree
    ✓ should revert with error tree doesnt exist (84ms)
    ✓ should get tree correctly (2050ms)
  getParent
    ✓ should revert with error tree doesnt exist (117ms)
    ✓ should get parent correctly (511ms)
  getTotalPurchase
    ✓ should revert with error tree doesnt exist (68ms)
    ✓ should get total purchase correctly (1023ms)
  getMemberPurchase
    ✓ should revert with error tree doesnt exist (43ms)
    ✓ should get member purchase correctly (1327ms)
  getNetworkPurchase
    ✓ should revert with error tree doesnt exist (116ms)
    ✓ should get network purchase correctly (1333ms)
  getDirectChildrenCount
    ✓ should revert with error tree doesnt exist (73ms)
    ✓ should get direct children count correctly (1492ms)

Contract: ManagedUpgradeable
  setManagementContract
    ✓ should revert with error caller is not the owner (272ms)
    ✓ should revert with error no contract (274ms)
    ✓ should set manegement contract address correctly (413ms)
  hasPermission
    ✓ should return false when user doesn't has permission (100ms)
    ✓ should return false when user doesn't has permission (380ms)
  requirePermission
    ✓ should return error when user doesn't has permission (289ms)
```

```
    ✓ should work correctly (314ms)
  canCallOnlyRegisteredContract
    ✓ should return error when user doesn't has permission (85ms)
    ✓ should work correctly (613ms)

Contract: ManagedUpgradeable
  setManagementContract
    ✓ should revert with error caller is not the owner (226ms)
    ✓ should revert with error no contract (225ms)
    ✓ should set manegement contract address correctly (705ms)
  hasPermission
    ✓ should return false when user doesn't has permission (73ms)
    ✓ should return false when user doesn't has permission (340ms)
  requirePermission
    ✓ should return error when user doesn't has permission (405ms)
    ✓ should work correctly (209ms)
  canCallOnlyRegisteredContract
    ✓ should return error when user doesn't has permission (83ms)
    ✓ should work correctly (677ms)

Contract: Management
  setKycWhitelists
    ✓ should revert with error access denied, caller doesn't has
      permission (236ms)
    ✓ should set kyc whitelist correctly (969ms)
  setLimitSetPermission
    ✓ should revert with error caller is not the owner (273ms)
    ✓ should set limit permission correctly (308ms)
  setPermission
    ✓ should revert with error access denied, caller is not the
      owner and don't have permission (293ms)
    ✓ should set permission correctly (321ms)
  setPermissions
    ✓ should revert with error caller is not the owner (255ms)
    ✓ should set permissions correctly (811ms)
  registerContract
    ✓ should revert with error caller is not the owner (319ms)
    ✓ should revert with error invalid address (244ms)
    ✓ should register contract correctly (796ms)
  isKYCPassed
    ✓ should revert with error time out (81ms)
```

```
    ✓ should return true if address have permission
      MANAGEMENT_WHITELISTED_KYC (69ms)
    ✓ should return true if user has MANAGEMENT_KYC_SIGNER
      permission (754ms)

Contract: WKRU
  name
    ✓ should return correct name (113ms)
  symbol
    ✓ should return correct symbol (143ms)
  totalSupply
    ✓ should return correct balance (613ms)
  decimals
    ✓ should return correct decimals (112ms)
  receive
    ✓ should deposit (614ms)
  deposit
    ✓ should revert with error amount is zero (261ms)
    ✓ should deposit (735ms)
    ✓ should catch event
  withdraw
    ✓ should revert with error more than max, amount to withdraw is
      more than user balance (333ms)
    ✓ should withdraw (992ms)
    ✓ should catch event
  approve
    ✓ should approve (980ms)
    ✓ should catch event
  transfer
    ✓ should revert with error more than max, amount to transfer
      is more than user balance (216ms)
    ✓ should transfer (1997ms)
    ✓ should catch event
  transferFrom
    ✓ should revert with error invalid address if sender is
      zero address (235ms)
    ✓ should revert with error invalid address if recipient is
      zero address (273ms)
    ✓ should revert with error amount is zero (267ms)
    ✓ should revert with error more than max, amount to transfer is
      more than user balance (314ms)
    ✓ should revert with error havent allocation (376ms)
```

```
    ✓ should transfer (957ms)
    ✓ should catch event

Contract: ManagementUpgradeable
  setKycWhitelists
    ✓ should revert with error access denied, caller doesn't has
      permission (304ms)
    ✓ should set kyc whitelist correctly (894ms)
  setLimitSetPermission
    ✓ should revert with error caller is not the owner (258ms)
    ✓ should set limit permission correctly (676ms)
  setPermission
    ✓ should revert with error access denied, caller is not
      the owner and don't have permission (355ms)
    ✓ should set permission correctly (358ms)
  setPermissions
    ✓ should revert with error caller is not the owner (711ms)
    ✓ should set permissions correctly (1142ms)
  registerContract
    ✓ should revert with error caller is not the owner (750ms)
    ✓ should revert with error invalid address (744ms)
    ✓ should register contract correctly (170ms)
  isKYCPassed
    ✓ should revert with error time out (138ms)
    ✓ should return true if address have permission
      MANAGEMENT_WHITELISTED_KYC (607ms)
    ✓ should return true if user has MANAGEMENT_KYC_SIGNER
      permission (437ms)

200 passing (3m)
```

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