SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Mimo Initiative ltd
Date: April 10, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Mimo Initiative Ltd
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	Exchange; ERC721; Rebase Token;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://mimo.capital/
Changelog	22.02.2023 - Initial Review 10.04.2023 - Second Review

Table of contents

Introduction 4
Scope 4
Severity Definitions 7
Executive Summary 8
Risks 9
Checked Items 10
System Overview 13
Findings 14
Critical 14
C01. Wrong Logic 14
High 14
H01. Wrong Logic 14
H02. Funds Lock 14
Medium 15
M01. Funds Lock 15
M02. Variable Name Contradiction 15
M03. Undocumented Feature 15
M04. Best Practice Violation 16
M06. Unbounded Variable 16
M09. Requirements Violation 16
M10. Unbounded Variable 16
M11. Inconsistent data 16
Low 17
L01. Redundant Code 17
L02. Redundant Override Keyword 17
L03. Unused Variables 18
L04. Redundant Read From Storage 18
L05. Solidity Style Guide 18
L06. Functionality Mismatch 18
L07. Inefficient Gas Model 18
L08. Best Practice Violation 19
Disclaimers 20

Introduction

Hacken OÜ (Consultant) was contracted by Mimo Initiative Ltd (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://github.com/mimo-capital/kuma-protocol
Commit	35952f041a9689d3731d130eb0ba8064cbf763af
Functional Requirements	https://github.com/mimo-capital/kuma-protocol/docs/README.md
Contracts	File: ./src/interfaces/IKBCToken.sol SHA3: 5fdfd5f23cd7a9536303c98eddffe66de0cea0442b57c739b8bc5856da44ec09 File: ./src/interfaces/IKIBToken.sol+ SHA3: 9316fb1a391de9a0a3492a35ccf7e311470f052b368ba33628a06e7f930c7782 File: ./src/interfaces/IKUMAAddressProvider.sol+ SHA3: f35a6a6ae2ac460ccef16da1e8644431888c955fee7004b52634a8a981f40d09 File: ./src/interfaces/IKUMAFeeCollector.sol+ SHA3: 0f469656485ae8462495170f6cf6286742094b39c38f527f1b90f5fe3ed0f370 File: ./src/interfaces/IKUMASwap.sol+ SHA3: 24e311702bf653218017ca09ca113fa1f7d7005e12bf9af6bccac16a868b1fdb File: ./src/interfaces/IMCAGRateFeed.sol+ SHA3: 77e1ecd5ce262f344b5e6c649d697e29cc7ac8975a431320ff6d7baa1f6fb7ab File: ./src/KBCToken.sol+ SHA3: 0a69debb6ee55c73ec1242e94fad390cbadff3035d43b67ec49095987e6cf6fb File: ./src/KIBToken.sol+ SHA3: 065dbc8e9a237b8d78fc73ea1ed8afa1594cfd61c363672e8948dc31ac4c4222 File: ./src/KUMAAccessController.sol+ SHA3: 38676a43817395baafe61f56fc40204350d78183382870f2d95bb99d04110174 File: ./src/KUMAAddressProvider.sol+ SHA3: 3fc4d4c4f7091fad24728ec919173465beb8942f276847065f97411afe31b819 File: ./src/KUMAFeeCollector.sol+ SHA3: a79ae75d1b75017b6881092912a7132f83863e9d9f7bcfd53ba3256d4165fcb7 File: ./src/KUMASwap.sol+ SHA3: 9a2da095f660a945440bb0338f278a10c170b68bcb586695cfb88bf545a4275c File: ./src/libraries/Errors.sol+

	SHA3: cb696f9c23b2af89e3da1db758340b8e3fb28b1a9a4ce111853977e585dbbdfe File: ./src/libraries/PercentageMath.sol+ SHA3: 2994326c8a8c777d1916146240e9476d01564ce558f0891055c59d9959adcc90 File: ./src/libraries/Roles.sol+ SHA3: ad84cb9878da2f12043ad770c54c7799345209baa6bf23422331703d68fb8a55 File: ./src/libraries/WadRayMath.sol+ SHA3: 33b953fcdc960ed1318a5d41b0cd23420b6ad708c44f63e7adf9e0a803adbbbd File: ./src/MCAGRateFeed.sol+ SHA3: 46286d46185a5cdc65494d9adbaa8002928990ff7c3ee47888bb64dfedf62848 File: ./script/PrintRoles.s.sol SHA3: ff560a7f4ad63dd5fc26d5ec40eb63e6a359d04dfa5b59afba251fd09a7e9bb4 Contracts in this repo have @mcag imports from https://github.com/hknio/mcag-contracts-03add80cb90a85e025e9d639.

Second review scope

Repository	https://github.com/mimo-capital/kuma-protocol
Commit	a0939464a1123f3a4175df7b6040a72e82dd8a27
Functional Requirements	https://github.com/mimo-capital/kuma-protocol/docs/README.md
Contracts	File: ./src/KBCToken.sol SHA3: b6391ab55311928426c2c55bd9ad23c74912ddc96b053b42d8f53f039127bfb3 File: ./src/KIBToken.sol SHA3: 3ca19858a0d072eddc5f5f2aaa82e28add463414941f24991ded41f7e1a37d89 File: ./src/KUMAAccessController.sol SHA3: 94d5a76d8979d17b2bae5ba5d8c1304cdbaa6f1548492c8143181de91df28a0a File: ./src/KUMAAddressProvider.sol SHA3: cf6808ca2374a517fd8e514d9bc2a2e9936b864d94ae6af149200ad1d7708142 File: ./src/KUMAFeeCollector.sol SHA3: 401860c93fd8faea8968f782ca1d5ca87c1c26646f6feee6de1f5b05b0791113 File: ./src/KUMASwap.sol SHA3: 30b6812d5edd94d67681f4eb6062e3e2f13038a840e6df9a612a9312294a6cbc File: ./src/MCAGRateFeed.sol SHA3: 8a369159124e52f9be80d8ffa1d07e68789b410fd7f62053056fbd79236ea8b0 File: ./src/interfaces/IKBCToken.sol SHA3: c4beab2beafc4843bcc66d4417310dd2d811e9fa44441c7346cd2992b4036009 File: ./src/interfaces/IKIBToken.sol SHA3: 45cb6766b0724643e5b68d65fb7a895805c485e40a0d1905a68ebe1755386431 File: ./src/interfaces/IKUMAAddressProvider.sol SHA3: d5c6dc7d821f5c040ccaa0a9d4b9f17e236c50c7bfab469a17701330cc515046 File: ./src/interfaces/IKUMAFeeCollector.sol SHA3: df42b076860d4d532361492743a39ee4dc441bd0c52b6f7511e3ca3a94b56f99

	File: ./src/interfaces/IKUMASwap.sol SHA3: ce7a41d89280ba0a3492eb50a42f30b5d9e77839cba0467d6980ce6161907218 File: ./src/interfaces/IMCAGRateFeed.sol SHA3: 585e876df6a4a9554f81f8bf25497f5ae5011b04e4f27515c947a3e96cc589f5 File: ./src/libraries/Errors.sol SHA3: ec4e8a2dcce47ac405b804b6e4b9af725726f977d2a5c9cd8c2235d30ad50e6c File: ./src/libraries/PercentageMath.sol SHA3: 2994326c8a8c777d1916146240e9476d01564ce558f0891055c59d9959adcc90 File: ./src/libraries/Roles.sol SHA3: 5f2933bcc623833bdcd2fa1c9d4876d117ad79bb6d881346f8e5f0ca35e43f5e File: ./src/libraries/WadRayMath.sol SHA3: 33b953fcdc960ed1318a5d41b0cd23420b6ad708c44f63e7adf9e0a803adbbbd

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Technical specification is not provided.
- Description of the development environment is not provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **98.68%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Some negative cases coverage is missing.

Security score

As a result of the audit, the code contains **3** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**.

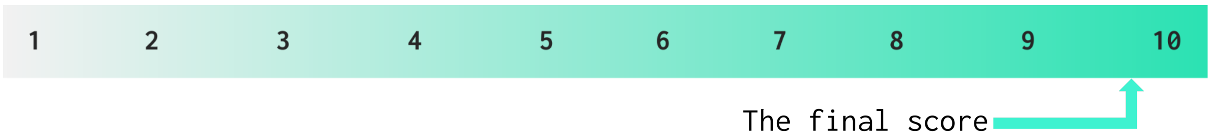

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
22 February 2023	6	11	2	2
10 April 2023	3	0	0	0

Risks

- The protocol is centralized.
- The calculation inside the protocol depends on the values received by another protocol that uses an oracle to get data off-chain.
- The stablecoin used by the KUMASWAP DAO to buy bonds can be changed without a timelock in deprecation mode. Users will receive a new stablecoin in exchange for their KIBToken in deprecation mode.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant

Token Supply Manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	Custom	Style guides and best practices should be followed.	Passed
Requirements Compliance	Custom	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	Custom	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	Custom	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	Custom	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	Custom	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

The KUMA Protocol is a blockchain-based platform that aims to tokenize bonds to generate low-risk, stable yields while utilizing the transparency and verifiability of smart contracts.
Mimo Capital AG (MCAG) is the centralized entity that holds the physical bonds and mints new KUMA Bonds NFTs every time a user buys a claim to the physical bonds off-chain. KUMA Bonds NFTs are ERC721 tokens that represent ownership of a physical bond and can be redeemed off-chain from MCAG for the market-rate bond value at any point. This part of the protocol is handled by a separate repo, MCAG-Contracts.
KUMA Interest Bearing Bond Tokens (KIBT) are rebase ERC20 tokens that represent a share of all bonds backed by the protocol, and each risk class has its own KIBToken.
The KUMA Interest Bearing Swap Contract holds all KUMA Bonds NFTs that back the KIBT, and users can sell and buy bonds from this contract. Each risk class has its own KIBT Swap Contract.
The protocol uses a centralized oracle to gather the current bond coupon for each risk class.
Keepers are centralized actors that monitor the KIBT and KUMASwap contracts to keep the KIBT yield up-to-date, while the KUMA DAO Access Controller is used for the decentralized contracts of the protocol.
The two main user scenarios are holding a KUMA Bonds NFT until maturity and redeeming it from MCAG, or selling the bond to the KUMASwap contract and receiving newly minted KIBT that can be used in the DeFi system.

Privileged roles

- KIBT_MINT_ROLE - Mints KIBTokens.
- KIBT_BURN_ROLE - Burns KIBTokens from any account without an allowance.
- KIBT_SWAP_PAUSE_ROLE - Pauses KUMASwap, which prevents all transfers, minting, and burning of KUMA Bonds NFTs.
- KIBT_SWAP_UNPAUSE_ROLE - Unpauses KUMASwap, which re-enables transfers, minting, and burning of KUMA Bonds NFTs after a pause.
- KIBT_SET_EPOCH_LENGTH_ROLE - Sets KIBToken epoch length.
- KUMA_MANAGER_ROLE - Sets configs of the protocol like sellBond fees, minGas in KIBTSwap , KUMAFeeCollector payees and shares, and contract addresses in the KUMAAddressProvider.
- KIBT_SWAP_CLAIM_ROLE - Claims the parent bonds of a clone bond in KIBTSwap.

Findings

■■■■ Critical

C01. Wrong Logic

The operations to compute the new base value divide the account value by *_cumulativeYield* instead of by *_previousEpochCumulativeYield* as correctly done in the mint*()* function.

Performing the division on the wrong *comulativeYield* variable will lead to a wrong computation of the base balances: two users involved in a transfer (or the user involved in a burn) will end up with a wrong balance. The *_totalBaseSupply* consistency would also be broken, as its value won't be reflected anymore in the *_baseBalances* mapping.

Path: ./src/KIBToken.sol : burn(), _transfer()

Recommendation: Replace *_cumulativeTield* with *_previousEpochCumulativeYield* in the lines computing the new base values in the *burn()* and *_transfer()* functions.

Status: Fixed (Revised commit: a093946)

■■■ High

H01. Wrong Logic

The *buyBondForStableCoin()* function does not enforce any rules on the paid amount, which could pose a risk to users' funds and the overall trust in the protocol.

It is acknowledged that this function is called by *KUMA_MANAGER_ROLE* after a DAO vote, but it is important to have consistent flow of operations in order to minimize the trust required by users in the protocol operators, as well as potential human errors. This is especially important in a critical situation such as refunding users after a Swap contract has been deprecated.

Path: ./src/KUMASwap.sol : buyBondForStableCoin()

Recommendation: It is recommended to enforce appropriate rules on the paid amount by the *buyBondForStableCoin()* function. For example, *KUMA_MANAGER_ROLE* could set a coupon-to-stablecoin ratio, and enforce it in this function.

Status: Mitigated (Note by customer: Intended behavior. Hard to enforce market value of bonds on-chain)

H02. Funds Lock

In the case of a deprecation mode users' funds are in a locked state until the DAO purchases the bonds.

Users are unable to buy back the bond or continue to use *KIBToken* as it was intended in a regular contract state. The funds remain effectively inaccessible until all the bonds are repurchased.

Path: ./src/KUMASwap.sol : redeemKIBT()

Recommendation: To avoid this scenario, the contract should keep track of the amount of bonds bought by the DAO, and unlock the relative portion of stablecoins for the users to redeem *KIBTokens*.

Status: Fixed (Revised commit: a093946)

■■ Medium

M01. Funds Lock

The *KUMASwap.sol* contract does not include a time lock mechanism that would enable the protocol managers to withdraw stablecoins that remain unclaimed by KIBT holders in a depreciation scenario.

Path: ./src/KUMASwap.sol

Recommendation: Implement a time lock to allow protocol owners to recover unused stablecoins by users to redeem *KIBTokens*.

Status: Mitigated (Note by customer: Intended behavior. KIBToken should always be backed either by a bond or by a stablecoin)

M02. Variable Name Contradiction

On line 198 a variable representing the bond value is called *bondFaceValue*.

The bond face value is a standard and established concept, semantically different from what this variable represents.

The same issue is replicated in the protocol documentation.

Path: ./src/KUMASwap.sol : buyBond()

Recommendation: Rename the variable to make it less confusing.

Status: Fixed (Revised commit: a093946)

M03. Undocumented Feature

The variable *maxCoupons* is arbitrarily set to *term / 30 days* in the constructor. This value significantly limits the operativity of the protocol, and has not been explained or motivated in the provided documentation.

Path: ./src/KUMASwap.sol : constructor()

Recommendation: Explain in the documentation the reasoning behind this limitation.

Status: Fixed (Revised commit: a093946)

M04. Best Practice Violation

The setter for the deprecated stablecoin does not implement a time lock.

Path: ./src/KUMASwap.sol : setDeprecationStableCoin()

Recommendation: The contract should implement a time lock for this setter, to show the user the change before it goes into effect.

Status: Mitigated (Note by customer: DAO responsibility)

M06. Unbounded Variable

The *KIBToken.sol* contract checks if the value of the variable *epochLength* is higher than 0, which is insufficient to fulfill the purpose of the variable, to allow keepers to have enough time to expire bonds.

Path: ./src/KIBToken.sol : constructor(), setEpochLength();

Recommendation: Implement reasonable boundaries for the *epochLength* variable, both in the constructor and in its setter.

Status: Fixed (Revised commit: a093946)

M09. Requirements Violation

Natspec requirements are violated for functions *redeem()* and *issueBond()* in the KBCToken contract.

Path: ./src/KBCToken.sol : redeem(), issueBond();

Recommendation: Enforce NatSpec requirements for these functions or adjust the Natspec.

Status: Fixed (Revised commit: a093946)

M10. Unbounded Variable

The fees' variables *_variableFee* and *_fixedFee* are unbounded in their setter *setFees()*.

Path: ./src/KUMASwap.sol : setFees()

Recommendation: Bound their value to a reasonable bound.

Status: Mitigated (Note by customer: checked in DAO functions)

M11. Inconsistent data

In case of activation of the deprecation mode getting for a swap, *KUMAAddressProvider* is still able to change its *KIBToken* address, possibly messing up the KIBT redemption process for users.

Path: ./src/KUMAAddressProvider.sol : setKIBToken()

Recommendation: Implement a process to make *KUMAAddressProvider* aware of the swap deprecation, and block the *setKIBToken()* function until a new swap gets deployed for that same risk class.

Status: Mitigated (Note by customer: DAO responsibility)

Low

L01. Redundant Code

The constructor is not using an *onlyValidAddress()* modifier, repeating the code instead.

Path: ./src/KUMAAddressProvider.sol : constructor()

Recommendation: Remove redundant code.

Status: Reported

L02. Redundant Override Keyword

Since *solidity 0.8.8*, a function that overrides only a single interface function does not require the *override* specifier.

The *override* keyword is used many times where it was not needed.

Path: ./src/KBCToken.sol : issueBond(), getBond(), redeem(), getTokenIdCounter();
./src/KIBToken.sol : setEpochLength(), refreshYield(), mint(), burn(), getYield(), getTotalBaseSupply(), getBaseBalance(), getEpochLength(), getCumulativeYield(), getUpdatedCumulativeYield();
./src/KUMAAddressProvider.sol : setKBCToken(), setRateFeed(), setKUMABondToken(), setKIBToken(), setKUMASwap(), setKUMAFeeCollector(), getKBCToken(), getRateFeed(), getKUMABondToken(), getKIBToken(), getKUMASwap(), getKUMAFeeCollector();
./src/KUMAFeeCollector.sol : release(), addPayee(), removePayee(), updatePayeeShare(), changePayees();
.src/KUMASwap.sol : sellBond(), buyBond(), buyBondForStableCoin(), claimBond(), redeemKIBT(), expireBond(), pause(), unpause(), setFees(), setDeprecatedStablecoin(), initializeDeprecationMode(), unintializeDeprecationMode(), enableDeprecationMode(), isDeprecationModeInitialized(), getDeprecationModeInitializedAt(), getVariableFee(), getFixedFee(), getDeprecationStablecoin(), getMinCoupon(), getCopouns(), getExpiredBonds(), getCouponIndex(), getBondReserve(), getBondIndex(), getCloneBond(), getCouponInventory(), isInReserve(), isExpired(), getBondBaseValue(), onERC721Received();
./src/MCAGRateFeed.sol : setOracle(), getRate(), getOracle(), minRateCoupon(), decimals();

Recommendation: Remove redundant code.

Status: Fixed (Revised commit: a093946)

L03. Unused Variables

The variables *_expirationDelay* and *MAX_YIELD* are never used.

Path: ./src/KUMASwap.sol; ./src/KIBToken.sol;

Recommendation: Remove unused variable.

Status: Fixed (Revised commit: a093946)

L04. Redundant Read From Storage

The storage variable *KUMAAddressProvider* has been loaded into memory in the *_KUMAAddressProvider* variable to be used for computations.

On line 156 the storage variable is read again, instead of using the memory copy.

Path: ./src/KUMASwap.sol : sellBond()

Recommendation: Remove redundant code.

Status: Fixed (Revised commit: a093946)

L05. Solidity Style Guide

The KUMAFeeCollector() function order does not follow the official guidelines.

Path: ./src/KYCToken.sol

Recommendation: Follow the official Solidity guidelines.

Status: Fixed (Revised commit: a093946)

L06. Functionality Mismatch

The function's name *getUpdatedCumulativeYield()* and its functionality do not match. This should be for the previous epoch cumulative yield, not the updated one.

Path: ./src/KIBToken.sol : getUpdatedCumulativeYield();

Recommendation: Rename the function.

Status: Reported

L07. Inefficient Gas Model

Functions *mint()*, *burn()*, *transfer()* do not return on *amount = 0*, possibly wasting the user's gas.

Path: ./src/KIBToken.sol : mint(), burn(), _transfer();

Recommendation: Check if the amount is equal to 0 before doing calculations inside the function and let these functions return in such cases.

Status: Reported

L08. Best Practice Violation

The setter for the deprecated stablecoin does not implement a time lock.

Path: ./src/KUMASwap.sol : setDeprecationStableCoin()

Recommendation: The contract should implement a time lock for this setter, to show the user the change before it goes into effect.

Status: Mitigated (Note by customer: DAO responsibility)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.