HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Kyber

Date: April 4th, 2021

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

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Kyber.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Governance
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/KyberNetwork/dao_sc/tree/katana
Commit	616906ABEED505F8D29186E147EB6736A914F49A
Timeline	19 MAR 2021 – 23 MAR 2021
Changelog	23 MAR 2021 – INITIAL AUDIT 4 APR 2021 – ADDING CUSTOMER NOTICE

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Kyber (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between March 19th, 2021 – March 23rd, 2021.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/KyberNetwork/dao_sc/tree/katana
Commit:
616906abeed505f8d29186e147eb6736a914f49a
Files:
/contracts/governance/KyberGovernance.sol
/contracts/governance/votingPowerStrategy/EpochVotingPowerStrategy.sol
/contracts/governance/executor/DefaultProposalValidator.sol
/contracts/governance/executor/DefaultExecutorWithTimelock.sol
/contracts/governance/executor/DefaultExecutor.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

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

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secure.

Insecure	Poor secured	Secured	Well-secured

You are here 

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

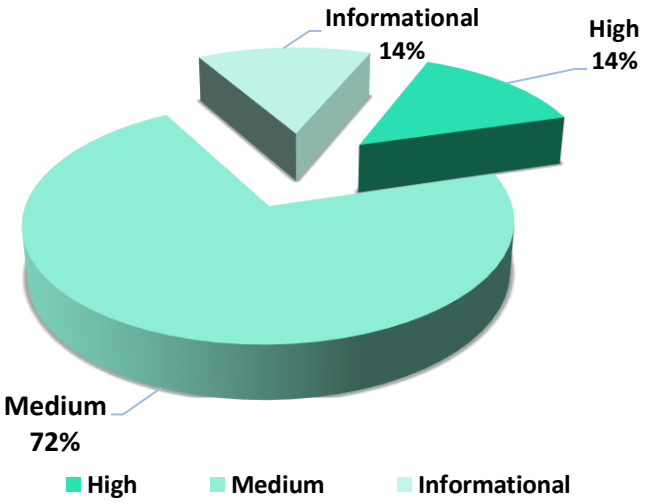
Security engineers found **1** high, **5** medium, and **1** informational issue during the audit. All issues are acknowledged and accepted by the Customer.

Notice:

1. The audit scope is limited to governance contracts. Its results may not be extrapolated to another contracts in the repository.
2. Writing universal contracts that can be used with different multiple implementations is considered a bad practice in solidity contracts. Such approach usually brings extra complexity and gas consumption.

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

Graph 1. The distribution of vulnerabilities after the first review.



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

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

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

AS-IS overview

KyberGovernance.sol

Description

KyberGovernance is a governance contract.

Imports

KyberGovernance has following imports:

- import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol'
- import {PermissionAdmin} from '@kyber.network/utils-sc/contracts/PermissionAdmin.sol'
- import {IKyberGovernance} from '../interfaces/governance/IKyberGovernance.sol'
- import {IExecutorWithTimelock} from '../interfaces/governance/IExecutorWithTimelock.sol'
- import {IVotingPowerStrategy} from '../interfaces/governance/IVotingPowerStrategy.sol'
- import {IProposalValidator} from '../interfaces/governance/IProposalValidator.sol'
- import {getChainId} from '../misc/Helpers.sol'

Inheritance

KyberGovernance is IKyberGovernance, PermissionAdmin.

Usages

KyberGovernance contract has following usages:

- SafeMath for uint256.

Structs

KyberGovernance contract has no custom data structures.

Enums

KyberGovernance contract has no custom enums.

Events

KyberGovernance contract has no custom events.

Modifiers
KyberGovernance has no custom modifiers.

Fields

KyberGovernance contract has following fields and constants:

- bytes32 public constant DOMAIN_TYPEHASH = keccak256('EIP712Domain(string name,uint256 chainId,address verifyingContract)');
- bytes32 public constant VOTE_EMITTED_TYPEHASH = keccak256('VoteEmitted(uint256 id,uint256 optionBitMask)');
- string public constant NAME = 'Kyber Governance';
- address private _daoOperator;
- uint256 private _proposalsCount;
- mapping(uint256 => Proposal) private _proposals;
- mapping(address => bool) private _authorizedExecutors;
- mapping(address => bool) private _authorizedVotingPowerStrategies;

Functions

KyberGovernance has following public and external functions:

- ***constructor***
  Description
  Initializes the contract. Sets adimin and dao operator addresses. Executors and votingStratages are also provided.
  Visibility
  public
  Input parameters
  - address admin
  - address daoOperator
  - address[] memory executors
  - address[] memory votingPowerStrategies

  Constraints
  None
  Events emit
  Emits the ExecutorAuthorized and VotingPowerStrategyAuthorized events.
  Output
  None
- ***createBinaryProposal***

Description
Creates a Binary Proposal.
Visibility
external
Input parameters
- IExecutorWithTimelock executor,
- IVotingPowerStrategy strategy,
- string[] memory options,
- uint256 startTime,
- uint256 endTime,
- string memory link

Constraints
- executor should be authorized.
- strategy should be authorized.
- executor should validate input params.

Events emit
Emits the BinaryProposalCreated event.
Output
uint256 proposalId

- ***createBinaryProposal***

Description
Creates a Binary Proposal. Creates a Generic Proposal. It only gets the winning option without any executions.
Visibility
external
Input parameters
- IExecutorWithTimelock executor
- IVotingPowerStrategy strategy
- BinaryProposalParams memory executionParams
- uint256 startTime
- uint256 endTime
- string memory link

Constraints
- executionParams should be provided.
- executionParams should be of the same length.
- executor should be authorized.
- strategy should be authorized.
- executor should validate input params.

Events emit
Emits the GenericProposalCreated event.

Output

uint256 proposalId

- ***cancel***

  Description

  Cancels a Proposal.In the currenct version is only callable by the _daoOperator. Though, if if the IProposalValidator implementation will allow, the function may be callable by anyone.

  Visibility

  external

  Input parameters

  - uint256 proposalId

  Constraints

  - Proposal should exist.
  - Should not be in a final state.

  Events emit

  Emits the ProposalCanceled event.

  Output

  None

- ***queue***

  Description

  Queue a proposal if it is succeeded.

  Visibility

  external

  Input parameters

  - uint256 proposalId

  Constraints

  - Proposal should exist.
  - Proposal should be Binary.
  - Should be in the Succeeded state.

  Events emit

  Emits the ProposalQueued event.

  Output

  None

- ***execute***

  Description

  Execute a proposal.

  Visibility

  external

  Input parameters

  - uint256 proposalId

Constraints

- Proposal should exist.
- Proposal should be Binary.
- Should be in the Queued state.

Events emit

Emits the ProposalExecuted event.

Output

None

- ***submitVote***

Description

Submit a vote for a proposal by message sender.

Visibility

external

Input parameters

- uint256 proposalId
- uint256 optionBitMask

Constraints

- Proposal should exist.
- Should be in the Active state.

Events emit

Emits the VoteEmittedevent.

Output

None

- ***submitVoteBySignature***

Description

Submit a vote for a proposal by signature.

Visibility

external

Input parameters

- uint256 proposalId
- uint256 optionBitMask
- uint8 v
- bytes32 r
- bytes32 s

Constraints

- Proposal should exist.
- Signature should be valid.
- Should be in the Active state.

Events emit

Emits the VoteEmittedevent.

Output
None

- ***handleVotingPowerChanged***
  Description
  Changes voting power of a voter.
  Visibility
  external
  Input parameters
  - uint256 proposalId
  - uint256 optionBitMask
  - uint8 v
  - bytes32 r
  - bytes32 s

  Constraints
  - Should be called from a strategy contract

  Events emit
  Emits the VotingPowerChanged.
  Output
  None
- ***transferDaoOperator***
  Description
  Transfers dao operator to another address. Should be called from a current dao operator.
- ***authorizeExecutors, unauthorizeExecutors, authorizeVotingPowerStrategies, unauthorizeVotingPowerStrategies***
  Description
  Admin functions to change corresponding contract parameters.
- ***isExecutorAuthorized, isVotingPowerStrategyAuthorized, getDaoOperator, getProposalsCount, getProposalById, getProposalVoteDataById, getVoteOnProposal, getProposalStat***
  Description
  Simple view functions.

EpochVotingPowerStrategy.sol

Description

EpochVotingPowerStrategy – Voting Power Strategy contract based on epoch mechanism.

Imports

EpochVotingPowerStrategy has following imports:

- import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol'
- import {IVotingPowerStrategy} from '../../interfaces/governance/IVotingPowerStrategy.sol'
- import {IKyberGovernance} from '../../interfaces/governance/IKyberGovernance.sol'
- import {IKyberStaking} from '../../interfaces/staking/IKyberStaking.sol'
- import {EpochUtils} from '../../misc/EpochUtils.sol'

Inheritance

EpochVotingPowerStrategy is IVotingPowerStrategy, EpochUtils.

Usages

EpochVotingPowerStrategy contract has following usages:

- SafeMath for uint256.

Structs

EpochVotingPowerStrategy contract has no custom data structures.

Enums

EpochVotingPowerStrategy contract has no custom enums.

Events

EpochVotingPowerStrategy contract has no custom events.

Modifiers

EpochVotingPowerStrategy has following custom modifiers:

- onlyStaking – check if a caller is staking.
- onlyGovernance – check if a caller is governance.

Fields

EpochVotingPowerStrategy contract has following fields and constants:

- uint256 public constant MAX_PROPOSAL_PER_EPOCH = 10
- IKyberStaking public immutable staking
- IKyberGovernance public immutable governance
- mapping(uint256 => uint256[]) internal epochProposals

Functions

EpochVotingPowerStrategy has following public and external functions:

- ***constructor***
  Description
  Initializes the contract. Sets governance and staking addresses.
  public
  Input parameters
  - IKyberGovernance _governance
  - IKyberStaking _staking

  Constraints
  None
  Events emit
  None
  Output
  None
- ***handleProposalCreation***
  Description
  Add a proposal ID to a current epoch.
  Input parameters
  - uint256 proposalId
  - uint256 startTime
  - uint256 – parameter exists but not used to be compliant with the interface.

  Constraints
  - Should only be called from a governance contract.

  Events emit
  None
  Output
  None
- ***handleProposalCancellation***
  Description
  Removes a proposal ID from a current epoch.
  Input parameters
  - uint256 proposalId

  Constraints
  - Should only be called from a governance contract.

  Events emit
  None
  Output

None

- ***handleProposalCancellation***
  Description
  Returns voter's voting power.
  Input parameters
  - address voter
  - uint256
  - uint256

  Constraints
  - Should only be called from a governance contract.

  Events emit
  None
  Output
  uint256 votingPower
- ***handleWithdrawal***
  Description
  Handle user withdraw from staking contract.
  Input parameters
  - address user
  - uint256 /*reduceAmount*/

  Constraints
  - Should only be called from a staking contract.

  Events emit
  None
  Output
- ***getVotingPower, validateProposalCreation, getMaxVotingPower, getListProposalIds***
  Description
  Simple view functions.

DefaultProposalValidator.sol

Description

DefaultProposalValidator is a contract with view or pure functions used to validate values or to retrieve data.

DefaultExecutorWithTimelock.sol

Description

DefaultExecutorWithTimelock is a contract that can queue, execute, cancel transactions voted by Governance.

Imports

DefaultExecutorWithTimelock has following imports:

- import {IExecutorWithTimelock} from '../../interfaces/governance/IExecutorWithTimelock.sol'
- import {IKyberGovernance} from '../../interfaces/governance/IKyberGovernance.sol'
- import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol'

Inheritance

DefaultExecutorWithTimelock is IExecutorWithTimelock.

Usages

DefaultExecutorWithTimelock contract has following usages:

- SafeMath for uint256.

Structs

DefaultExecutorWithTimelock contract has no custom data structures.

Enums

DefaultExecutorWithTimelock contract has no custom enums.

Events

DefaultExecutorWithTimelock contract has no custom events.

Modifiers

DefaultExecutorWithTimelock has following modifiers:

- onlyAdmin – check for a corresponding caller.
- onlyTimelock – check for a corresponding caller.
- onlyPendingAdmin – check for a corresponding caller.

Fields

DefaultExecutorWithTimelock contract has following fields and constants:

- uint256 public immutable override GRACE_PERIOD
- uint256 public immutable override MINIMUM_DELAY
- uint256 public immutable override MAXIMUM_DELAY
- address private _admin
- address private _pendingAdmin
- uint256 private _delay
- mapping(bytes32 => bool) private _queuedTransactions

Functions

DefaultExecutorWithTimelock has following public and external functions:

- ***constructor***
  Description
  Initializes the contract. Sets admin, delay, grace period for queued txs, MINIMUM_DELAY and MAXIMUM_DELAY values.
  Visibility
  public
  Input parameters
  - address admin
  - uint256 delay
  - uint256 gracePeriod
  - uint256 minimumDelay
  - uint256 maximumDelay

  Constraints
  - delay should be between minimumDelay and maximumDelay.

  Events emit
  Emits the NewDelay and NewAdmin events.
  Output
  None
- ***setDelay, setPendingAdmin***
  Description
  Allows the contract to change corresponding parameters.
- ***acceptAdmin***
  Description
  Allows pending admin to accept his permissions.
- ***queueTransaction***
  Description
  Queues a transaction for execution.

Visibility
public
Input parameters
- address target
- uint256 value
- string memory signature
- bytes memory data
- uint256 executionTime
- bool withDelegatecall

Constraints
- Could only be called from the admin account.
- executionTime should exceed or be equal to a current time + delay.

Events emit
Emits the QueuedAction event.
Output
bytes32 – action hash.

- ***cancelTransaction***

Description
Cancel a transaction.
Visibility
public
Input parameters
- address target
- uint256 value
- string memory signature
- bytes memory data
- uint256 executionTime
- bool withDelegatecall

Constraints
- Could only be called from the admin account.

Events emit
Emits the CancelledAction event.
Output
bytes32 – action hash.

- ***executeTransaction***

Description
Execute a transaction.
Visibility
public
Input parameters

- address target
- uint256 value
- string memory signature
- bytes memory data
- uint256 executionTime
- bool withDelegatecall

Constraints

- Could only be called from the admin account.
- Transaction should be queued.
- Current time should be after execution time and before grace period pass.
- Transaction should be sent successfully.

Events emit

Emits the ExecutedAction event.

Output

bytes memory

- ***getAdmin, getPendingAdmin, getDelay, isActionQueued, isProposalOverGracePeriod***

  Description

  Simple view functions.

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

Audit overview

Critical

No critical issues were found.

High

1. Admin can be set by voting. This can lead to lost of admin permissions by owners.

   Contract: DefaultExecutorWithTimelock.sol

   Recommendation: allow admin change only for current admin.

   Customer notice: By design, the admin of all executor timelocks will be the KyberGovernance contract, thus, the only way to change the admin role is via a proposal. That's the reason why we require the sender of updating admin to be the timelock contract itself.

Medium

1. Reviewed contracts have high cohesion. This leads to increased gas consumption because of regular external calls between contracts.

   Contracts: KyberGovernance.sol, DefaultProposalValidator.sol, DefaultExecutorWithTimelock.sol, EpochVotingPowerStrategy.sol.

   Examples:

   - handleProposalCancellation function of the EpochVotingPowerStrategy can be called only from a governance contract and calls the same governance to retrieve some data.

   Recommendation: we recommend decreasing number of external calls between those contracts as much as it's possible.

   Customer notice: There are multiple voting power strategies that can be implemented in the future and the governance contract doesn't know which data each voting power will need to update their logic., Thus, we pass only the proposal id when calling handleProposalCancellation. We decided that having the flexibility for new voting power strategy implementations was more crucial than having lower gas consumption.

2. MIN_VOTING_DURATION, MAX_VOTING_OPTIONS, VOTE_DIFFERENTIAL, MINIMUM_QUORUM constants are not validated for lower and upper limits when initialized..

   Contract: DefaultProposalValidator.sol

   Recommendation: set up upper and lower limits.

   Customer notice: These (constant) configurations will be carefully decided before we deploy all contracts, thus, we don't really want to add any checks for the limitation of each value. If the KyberGovernance wants to change any configuration, there will be a proposal to vote for an upgrade.

3. GRACE_PERIOD, MINIMUM_DELAY, MAXIMUM_DELAY constants are not validated for lower and upper limits when initialized.

   Contract: DefaultExecutorWithTimelock.sol

   Recommendation: set up upper and lower limits.

   Customer notice: These (constant) configurations will be carefully decided before we deploy all contracts, thus, we don't really want to add any checks for the limitation of each value. If the KyberGovernance wants to change any configuration, there will be a proposal to vote for an upgrade.

4. DefaultExecutorWithTimelock and DefaultProposalValidator are both used by the KyberGovernance as IExecutorWithTimelock. Though, those contracts implement different interfaces.

   Contract: KyberGovernance.sol

   Recommendation: use separate addresses IExecutorWithTimelock and IProposalValidator contracts in the KyberGovernance contract.

   Customer notice: As the ProposalValidator and ExecutorWithTimelock contracts are tightly coupled, it makes sense to have a contract to inherit both of these contracts and use that as the Executor instead of keeping them separate.

5. Function may fail due to block gas limit if number of epochProposals of a current epoch is big enough. This may lead to fails of staking functions.

Contract: EpochVotingPowerStrategy.sol

Function: handleWithdrawal

Recommendation: limit number of proposals for each epoch.

Customer notice: In the validateProposalCreation function of EpochVotingPowerStrategy, we already validate that the number of proposals for each epoch must not be greater than MAX_PROPOSAL_PER_EPOCH, which is likely to be at most 10 proposals per epoch.

Low

No low severity issues were found.

Informational / Code style / Best Practice

1. Some code style issues were found by static code analyzers.

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

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **1** high, **5** medium, and **1** informational issue during the audit. All issues are acknowledged and accepted by the Customer.

Notice:

1. The audit scope is limited to governance contracts. Its results may not be extrapolated to another contracts in the repository.
2. Writing universal contracts that can be used with different multiple implementations is considered a bad practice in solidity contracts. Such approach usually brings extra complexity and gas consumption.

Violations in the following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Costly loops	▪ The code may fail due to block gas limit because of data processing in a loop.
	▪ Access Control & Authorization	▪ Access to the Timelock contract can be lost
	▪ Style guide violation	▪ Some code style issues were found
	▪ Architecture review	▪ Architecture of the reviewed contracts is not optimal.

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

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. 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 security of smart contracts.

Technical Disclaimer

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