HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: KeyFi

Date: November 22nd, 2020

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 KeyFi
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, Multisig Timelock, Token factory,
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/KEYFIAI/keyfi-token
Commit	HTTPS://GITHUB.COM/KEYFIAI/KEYFI-TOKEN/COMMIT/DC47D232C3F22F2559654E47BD3DF7EE91F48974
Deployed contract	
Timeline	10 NOV 2020 – 22 NOV 2020
Changelog	12 NOV 2020 – INITIAL AUDIT 22 NOV 2020 – SECONDARY AUDIT

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 KeyFi (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 November 10th, 2020 – November 12th, 2020. The secondary review conducted between November 20th, 2020 – November 22nd, 2020

Scope

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

Repository: https://github.com/KEYFIAI/keyfi-token
Commit: HTTPS://GITHUB.COM/KEYFIAI/KEYFI-TOKEN/COMMIT/DC47D232C3F22F2559654E47BD3DF7EE91F48974
Files:

- KeyfiToken.sol
- KeyfiTokenFactory.sol – removed before the second audit.
- Migrations.sol
- MultisigTimelock.sol – removed before the second audit.
- RewardPool.sol
- Whitelist.sol – added before the second audit.
- GovernorAlpha.sol – added before the second audit.

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 ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secure and can be used in the production.

Insecure	Poor secured	Secured	Well-secured

You are 

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.

As a result of the **first audit**, security engineers found 1 critical, 3 high and 4 medium severity issues during the audit.

After the **second audit**, the contract contains 2 medium, 2 low and 1 lowest severity issues.

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.

AS-IS overview

KeyfiToken.sol

Description

KeyfiToken is an ERC-20 token with voting and minting functionality. Keyfi token has following parameters:

- Name: Keyfi Token
- Symbol: KEYFI
- Decimals: 18

Imports

KeyfiToken contract has following imports:

- Ownable
- IERC20

Inheritance

KeyfiToken contract is IERC20, Ownable.

Usages

KeyfiToken contract has no custom usages.

Structs

KeyfiToken contract has following data structures:

- struct Checkpoint

Enums

KeyfiToken contract has no custom enums.

Events

KeyfiToken contract has following events:

- event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);
- event DelegateVotesChanged(address indexed delegate, uint256 previousBalance, uint256 newBalance);

- event Transfer(address indexed from, address indexed to, uint256 amount);
- event Approval(address indexed owner, address indexed spender, uint256 amount);

Modifiers

KeyfiToken has no custom modifiers.

Fields

KeyfiToken contract has following fields and constants:

- string public constant name = "Keyfi Token";
- string public constant symbol = "KEYFI";
- uint8 public constant decimals = 18;
- uint256 public override totalSupply = 0;
- mapping (address => mapping (address => uint256)) internal allowances;
- mapping (address => uint256) internal balances;
- mapping (address => address) public delegates;
- mapping (address => mapping (uint256 => Checkpoint)) public checkpoints; mapping (address => uint256) public numCheckpoints;
- bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
- bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");
- mapping (address => uint) public nonces;

Functions

KeyfiToken has following public functions:

- ***mint***
  Description
  Mint tokens
  Visibility
  public
  Input parameters
  - address _to
  - uint256 _amount

  Constraints

- Can only be called by the owner.

Events emit

Emits `Transfer` event.

Output

None

- ***allowance***

Description

Get the number of tokens `spender` is approved to spend on behalf of `account`

Visibility

external view

Input parameters

- address account
- address spender

Constraints

None

Events emit

None

Output

- uint256 — The number of tokens approved.

- ***approve***

Description

Approve `spender` to transfer up to `amount` from `src`

Visibility

external

Input parameters

- address spender
- uint256 amount

Constraints

None

Events emit

None

Output

- bool — Whether or not the approval succeeded.

- ***approve***

Description

Approve `spender` to transfer up to `amount` from `src`.

Visibility

external

Input parameters

- address spender
- uint256 amount

Constraints
None
Events emit
Emits `Approval` event.
Output

- bool — Whether or not the approval succeeded.

- ***balanceOf***
  Description
  Get the number of tokens held by the `account`.
  Visibility
  external view
  Input parameters
  - address account

  Constraints
  None
  Events emit
  None
  Output
  - uint256 — The number of tokens held.
- ***transfer***
  Description
  Transfer `amount` tokens from `msg.sender` to `dst`.
  Visibility
  external
  Input parameters
  - address dst
  - uint256 amount

  Constraints
  - A sender should have enough tokens.

  Events emit
  Emits `Transfer` event.
  Output
  - bool — Whether or not the transfer succeeded.
- ***transferFrom***
  Description
  Transfer `amount` tokens from `src` to `dst`.
  Visibility
  external

Input parameters
- address src
- address dst
- uint256 amount

Constraints
- A message sender should have allowance to transfer tokens from src.

Events emit
Emits `Approval` and `Transfer` event.
Output
- bool — Whether or not the transfer succeeded.

- ***delegate***

Description
Delegate votes from `msg.sender` to `delegatee`
Visibility
public
Input parameters
- address delegatee

Constraints
None
Events emit
Emits `DelegateChanged` event.
Output
None

- ***delegateBySig***

Description
Delegates votes from signatory to `delegatee`.
Visibility
public
Input parameters
- address delegate
- uint256 nonce
- uint256 expiry
- uint8 v
- bytes32 r
- bytes32 s

Constraints
None
Events emit
Emits `DelegateChanged` event.

Output
None

- ***getCurrentVotes***
  Description
  Get current votes balance for `account`.
  Visibility
  external view
  Input parameters
  - address account

  Constraints
  None
  Events emit
  Emits `DelegateChanged` event.
  Output
  - uint256 — number of current votes for `account`.

- ***getCurrentVotes***
  Description
  Get current votes balance for `account`.
  Visibility
  external view
  Input parameters
  - address account

  Constraints
  None
  Events emit
  None
  Output
  - uint256 — number of current votes for `account`.

- ***getPriorVotes***
  Description
  Determine the prior number of votes for an account as of a block number.
  Visibility
  public view
  Input parameters
  - address account
  - uint256 blockNumber

  Constraints
  None

Events emit
None
Output
- uint256 — number of votes the account had as of the given block.

Whitelist.sol

Description

Whitelist is a contract that provides whitelist functionality.

Imports

Whitelist contract has following imports:

- AccessControl

Inheritance

Whitelist contract is AccessControl.

Usages

Whitelist contract has no custom usages.

Structs

Whitelist contract has no custom data structures.

Enums

Whitelist contract has no custom enums.

Events

Whitelist contract has no custom events.

Modifiers

Whitelist has following modifiers:

- onlyWhitelistAdmin – checks whether a caller is whitelist admin.
- onlyWhitelisted – checks whether a caller is whitelisted.

Fields

Whitelist contract has following fields and constants:

- bytes32 public constant WHITELIST_ADMIN = keccak256("WHITELIST_ADMIN");
- bytes32 public constant WHITELISTED = keccak256("WHITELISTED");

Functions

Whitelist has following public functions:

- ***constructor***
  Description
  Inits the contract. Sets whitelist admin role to a message sender
  Visibility
  public
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  None
- ***addWhitelistAdmin***
  Description
  Adds new whitelist admin
  Visibility
  public
  Input parameters
  - address account

  Constraints
  - Can only be called by the whitelist admin.

  Events emit
  None
  Output
  None
- ***removeWhitelistAdmin***
  Description
  Remove a whitelist admin
  Visibility
  public

Input parameters
- address account

Constraints
- Can only be called by the whitelist admin.

Events emit
None
Output
None

- ***addWhitelisted***

Description
Add an `address` to the whitelist.
Visibility
public
Input parameters
- address account

Constraints
- Can only be called by the whitelist admin.

Events emit
None
Output
None

- ***removeWhitelisted***

Description
Remove an `address` from the whitelist.
Visibility
public
Input parameters
- address account

Constraints
- Can only be called by the whitelist admin.

Events emit
None
Output
None

- ***removeWhitelisted***

Description
Check whether an `address` is whitelisted.
Visibility
public view
Input parameters

- address account

Constraints
None
Events emit
None
Output
None

- ***isWhitelistAdmin***
Description
Check whether an `address` is the whitelist admin.
Visibility
public view
Input parameters
  - address account

Constraints
None
Events emit
None
Output
None

KeyfiTokenFactory.sol (Removed before the second audit)

Description

KeyfiTokenFactory is a factory used to deploy the KeyfiToken.

Imports

KeyfiTokenFactory contract has following imports:

- KeyfiToken
- Whitelist
- RewardPool
- TokenTimelock – from the OpenZeppelin.
- IERC20 – from the OpenZeppelin.

Inheritance

KeyfiTokenFactory does not inherit any contracts.

Usages

KeyfiTokenFactory contract has following usages:

- using *SafeERC20 for KeyfiToken;*

Structs

KeyfiTokenFactory contract has no custom data structures.

Enums

KeyfiTokenFactory contract has no custom enums.

Events

KeyfiTokenFactory contract has following events:

- RewardPoolDeployed(address pool, address owner)

Modifiers

KeyfiTokenFactory has no custom modifiers.

Fields

KeyfiTokenFactory contract has following fields and constants:

- KeyfiToken public token;
- RewardPool public pool;
- address public community;
- TokenTimelock public teamTimelock1;
- TokenTimelock public teamTimelock2;
- TokenTimelock public teamTimelock3;
- TokenTimelock public teamTimelock4;;

Functions

KeyfiTokenFactory has following public functions:

- ***constructor***
  Description
  Initializes contract and deploys *KeyfiToken and 4 timelocks for the team.* Mints 10000000 tokens. Transfers 400000 tokens to the team address, 400000 tokens to each of 4 timeloks, and 250000 tokens to the airdrop contract. Transfers ownership of the token to the community address.

  Visibility
  public

Input parameters

- address team,
- address community,
- address airdrop

Constraints

None

Events emit

Emits `KeyfiTokenFactoryDeployed` event.

Output

None

- ***deployRewardPool***

  Description

  Deploys a reward pool contract and transfers 6250000 to it. Transfers ownership of the reward pool to the community.

  Visibility

  public

  Input parameters

  - address team
  - address community
  - address airdrop

  Constraints

  None

  Events emit

  Emits `RewardPoolDeployed` event.

  Output

  None

MultisigTimelock.sol (Removed before the second audit)

Description

MultisigTimelock a multisig wallet with timelock functionality.

Imports

MultisigTimelock contract has no imports.

Inheritance

MultisigTimelock does not inherit any contracts.

Usages

MultisigTimelock contract has no usages.

Structs

MultisigTimelock contract has following data structures:

- Transaction

Enums

MultisigTimelock contract has no custom enums.

Events

MultisigTimelock contract has following events:

- event Confirmation(address indexed sender, uint256 indexed transactionId);
- event Revocation(address indexed sender, uint256 indexed transactionId);
- event Submission(uint256 indexed transactionId);
- event Execution(uint256 indexed transactionId);
- event ExecutionFailure(uint256 indexed transactionId);
- event Deposit(address indexed sender, uint256 value);
- event OwnerAddition(address indexed owner);
- event OwnerRemoval(address indexed owner);
- event RequirementChange(uint256 required);
- event UnlockTimeSet(uint256 indexed transactionId, uint256 confirmationTime);
- event LockSecondsChange(uint256 lockSeconds);

Modifiers

MultisigTimelock has following modifiers:

- onlyWallet() – checks whether a call was made from the contract itself.
- ownerDoesNotExist(address owner) – checks whether an owner does not exist.
- ownerExists(address owner) – checks whether an owner exists.
- transactionExists(uint256 transactionId) – checks whether a transaction with `transactionId` exists.
- confirmed(uint256 transactionId, address owner) – checks whether a transaction with `transactionId` is confirmed by `owner`.

- notConfirmed(uint256 transactionId, address owner) – checks whether a transaction with `transactionId` is not confirmed by `owner`.
- modifier notExecuted(uint256 transactionId) – checks whether a transaction with `transactionId` is not yet executed.
- modifier notNull(address _address) – checks whether an `_address` is not 0 address.
- modifier validRequirement(uint256 ownerCount, uint256 _required) – checks whether the `ownerCount` is greater or equal to `required`, whether the `ownerCount` does not exceed `MAX_OWNER_COUNT`, and whether the `ownerCount` and the `required` are both greater than 0.

Fields

MultisigTimelock contract has following fields and constants:

- uint256 constant public MAX_OWNER_COUNT = 50;
- uint256 public lockSeconds = 259200; // 3 days
- mapping (uint256 => Transaction) public transactions;
- mapping (uint256 => mapping (address => bool)) public confirmations;
- mapping (address => bool) public isOwner;
- mapping (uint256 => uint256) public unlockTimes;
- address[] public owners;
- uint256 public required;
- uint256 public transactionCount;

Functions

MultisigTimelock has following public functions:

- ***constructor***

  Description

  Init the contract and sets initial owners and a number of required confirmations.

  Visibility

  public

  Input parameters

  - address[] memory _owners
  - uint256 _required

  Constraints

  - validRequirement modifier

  Events emit

  None

  Output

None

- ***receive()***
  Description
  Allows to deposit ETH
- ***addOwner***
  Description
  Adds a new owner.
  Visibility
  external
  Input parameters
  - address owner

  Constraints
  - onlyWallet modifier
  - ownerDoesNotExist modifier
  - notNull modifier
  - validRequirement modifier

  Events emit
  Emits OwnerAddition event.
  Output
  None
- ***removeOwner***
  Description
  Removes an owner.
  Visibility
  external
  Input parameters
  - address owner

  Constraints
  - onlyWallet modifier
  - ownerExists modifier

  Events emit
  Emits OwnerRemoval event.
  Output
  None
- ***replaceOwner***
  Description
  Replace an owner with a new owner.
  Visibility
  external
  Input parameters

- address owner
- address newOwner

Constraints

- onlyWallet modifier
- ownerExists modifier
- ownerDoesNotExist modifier

Events emit

Emits OwnerRemoval and OwnerAddition events.

Output

None

- ***changeRequirement***

  Description

  Allows to change the number of required confirmations.

  Visibility

  external

  Input parameters

  - uint256 _required

  Constraints

  - onlyWallet modifier
  - validRequirement modifier

  Events emit

  Emits RequirementChange event.

  Output

  None

- ***changeLockSeconds***

  Description

  Changes the duration of the time lock for transactions.

  Visibility

  external

  Input parameters

  - uint256 _lockSeconds

  Constraints

  - onlyWallet modifier

  Events emit

  Emits LockSecondsChange event.

  Output

  None

- ***submitTransaction***

  Description

Submit a new transaction and confirms it on behalf of the message sender.

Visibility
external
Input parameters
- address destination
- uint256 value
- bytes calldata data

Constraints
- ownerExists modifier
- notNull modifier

Events emit
Emits Submission and Confirmation events.
Output
- uint256 transactionId – the transaction id.

- ***confirmTransaction***

Description
Confirms a transaction.
Visibility
public
Input parameters
- uint256 transactionId

Constraints
- ownerExists modifier
- transactionExists modifier
- notConfirmed modifier

Events emit
Emits Confirmation and UnlockTimeSet events.
Output
None

- ***revokeConfirmation***

Description
Revokes confirmation.
Visibility
external
Input parameters
- uint256 transactionId

Constraints
- ownerExists modifier
- confirmed modifier

- notExecuted modifier

Events emit

Emits Revocation event.

Output

None

- ***executeTransaction***

Description

Executes a transaction.

Visibility

external

Input parameters

- uint256 transactionId

Constraints

- ownerExists modifier
- notExecuted modifier
- the transaction should be confirmed and unlockTime should pass after its latest confirmation

Events emit

Emits Revocation event.

Output

None

- ***isConfirmed***

Description

Returns the confirmation status of a transaction.

Visibility

public view

Input parameters

- uint256 transactionId

Constraints

None

Events emit

None

Output

- bool — confirmation status

- ***getConfirmationCount***

Description

Returns number of confirmations of a transaction.

Visibility

external view

Input parameters

- uint256 transactionId

Constraints
None
Events emit
None
Output
- uint256 count — confirmations number.

- ***getTransactionCount***
Description
Returns total number of transactions that match provided filters.
Visibility
external view
Input parameters
  - bool pending
  - bool executed
Constraints
None
Events emit
None
Output
  - uint256 count — number of transactions.
- ***getOwners***
Description
Returns list of owners.
Visibility
external view
Input parameters
None
Constraints
None
Events emit
None
Output
  - address[] memory – list of owners.
- ***getConfirmations***
Description
Returns a list of owners whou confirmed a transaction
Visibility
external view
Input parameters

- uint256 transactionId

Constraints

None

Events emit

None

Output

- address[] memory _confirmations – list of those who confirmed

RewardPool.sol

Description

RewardPool is a staking contract with rewards in reward tokens.

Imports

RewardPool contract has following imports:

- IERC20
- SafeERC20
- SafeMath
- Ownable

Inheritance

RewardPool is Ownable

Usages

RewardPool contract following usages:

- SafeMath for uint256;
- SafeERC20 for IERC20;

Structs

RewardPool contract has following data structures:

- UserInfo
- StakingToken
- TokenIndex

Enums

RewardPool contract has no custom enums.

Events

RewardPool contract has following events:

- event TokenAdded(address indexed token, uint256 allocPoints);
- event TokenRemoved(address indexed token);
- event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
- event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
- event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

Modifiers

RewardPool has no custom modifiers.

Fields

RewardPool contract has following fields and constants:

- IERC20 public rewardToken – reward token.
- uint256 public bonusEndBlock – block number when bonus reward period ends.
- uint256 public rewardPerBlock – reward tokens distributed per block.
- uint256 public bonusMultiplier = 2; – bonus muliplier for early users.
- StakingToken[] public stakingTokens – info of each pool.
- mapping(address => TokenIndex) public stakingTokenIndexes – token indexes.
- mapping (uint256 => mapping (address => UserInfo)) public userInfo – Info of each user that stakes tokens.
- uint256 public totalAllocPoint = 0 – Total allocation points. Must be the sum of all allocation points in all pools.
- uint256 public startBlock — the block number when rewards start.
- Whitelist public whitelist – the whitelist contract.

Functions

RewardPool has following public functions:

- ***constructor***
  Description
  Inits the contract and sets default parameters.
  Visibility

public

Input parameters

- IERC20 _rewardToken
- uint256 _rewardPerBlock
- uint256 _startBlock
- uint256 _bonusEndBlock
- uint8 _bonusMultiplier
- Whitelist _whitelist

Constraints

None

Events emit

None

Output

None

- ***stakingTokensCount***

  Description

  Returns a number of staking tokens.

  Visibility

  external view

  Input parameters

  None

  Constraints

  None

  Events emit

  None

  Output

  - uint256 – number of staking tokens.

- ***addStakingToken***

  Description

  Adds a token to the list of allowed staking tokens.

  Visibility

  public

  Input parameters

  - uint256 _allocPoint — the "weight" of this token in relation to other allowed tokens.
  - IERC20 _stakingToken — the token to be added.

  Constraints

  - Can only be called by the owner.
  - The `_stakingToken` should not be equal to the `rewardToken`
  - The `_stakingToken` should not be added yet.

Events emit
Emits the `TokenAdded` event.
Output
None

- ***removeStakingToken (Removed before the second audit)***
  Description
  Removes a staking token from the list of allowed tokens.
  Visibility
  public
  Input parameters
  - IERC20 _stakingToken — The token to be removed.

  Constraints
  - Can only be called by the owner.
  - The `_stakingToken` should be added.

  Events emit
  Emits the `TokenRemoved` event.
  Output
  None
- ***set***
  Description
  Changes the weight allocation for a particular token.
  Visibility
  public
  Input parameters
  - IERC20 _stakingToken — the token to be removed.
  - uint256 _allocPoint — new value of allocPoint.

  Constraints
  - Can only be called by the owner.
  - The `_stakingToken` should be added.

  Events emit
  None
  Output
  None
- ***getMultiplier***
  Description
  Returns multiplier factor for possible bonuses within a period.
  Visibility
  public view
  Input parameters
  - uint256 _from — starting block

- uint256 _to — last block of the period

Constraints

None

Events emit

None

Output

None

- ***getMultiplier***

Description

Returns multiplier factor for possible bonuses within a period.

Visibility

public view

Input parameters

- uint256 _from — starting block
- uint256 _to — last block of the period

Constraints

None

Events emit

None

Output

- uint256 – multiplier factor.

- ***pendingReward***

Description

Calculates pending reward for a given staking token and a user. Only a whitelisted address can have pending rewards.

Visibility

external view

Input parameters

- IERC20 _token
- address _user

Constraints

- A token should be set.

Events emit

None

Output

- uint256 – pending reward.

- ***massUpdateTokens***

Description

Invokes a checkpoint update on all staking tokens in the list.

Visibility

public
Input parameters
None
Constraints
Non
Events emit
None
Output
None

- ***checkpoint***
  Description
  Calculates all reward rates for a specified token since last checkpoint.
  Visibility
  public
  Input parameters
  - uint256 _pid – a token id.

  Constraints
  - Token with `_pid` should exist.

  Events emit
  None
  Output
  None
- ***deposit***
  Description
  Deposit `_amount` into a given `_token` pool.
  Visibility
  public
  Input parameters
  - IERC20 _token — the staking token to be deposited.
  - uint256 _amount — The amount of tokens to be staked.

  Constraints
  - `_token` should be added.
  - A `msg.sender` should be whitelisted.

  Events emit
  Emits `Deposit` event.
  Output
  None
- ***withdraw***
  Description

Withdraw `_amount` of a given staking token. A reward will not be withdrawn if a `msg.sender` sender is not whitelisted

Visibility

public

Input parameters

- IERC20 _token — the staking token to be withdrawn.
- uint256 _amount — the amount of tokens to be withdrawn.

Constraints

- A `_token` should be added.
- A message sender should have at least `_amount` of tokens to be deposited earlier.

Events emit

Emits `Withdraw` event.

Output

None

- ***emergencyWithdraw***

Description

Withdraw all specified staked tokens without a reward.

Visibility

public

Input parameters

- IERC20 _token — the staking token to be withdrawn.

Constraints

- A `_token` should be added.

Events emit

Emits `EmergencyWithdraw` event.

Output

None

- ***rewardBlocksLeft***

Description

Calculate remaining blocks left according to current reward supply and rate. **Visibility**

public view

Input parameters

None

Constraints

None

Events emit

None

Output

- uint256 – remaining blocks.

- ***adminWithdrawReward (removed)***
  Description
  Allows owner to withdraw any amount of reward tokens.
  Visibility
  public
  Input parameters
  - uint256 amount — The amount of tokens to be withdrawn

  Constraints
  - Can only be called by the owner.

  Events emit
  None
  Output
  None

GovernorAlpha.sol

Description

GovernorAlpha allows to vote for specific actions using KeyFi token votes.

Imports

GovernorAlpha has no imports.

Inheritance

GovernorAlpha does not inherit anything.

Usages

GovernorAlpha contract has no usages.

Structs

GovernorAlpha contract has following data structures:

- Proposal
- Receipt

Enums

GovernorAlpha contract has following enums:

- ProposalState

Events

GovernorAlpha contract has following events:

- event ProposalCreated(uint id, address proposer, address[] targets, uint[] values, string[] signatures, bytes[] calldatas, uint startBlock, uint endBlock, string description);
- event VoteCast(address voter, uint proposalId, bool support, uint votes);
- event ProposalCanceled(uint id);
- event ProposalQueued(uint id, uint eta);
- event ProposalExecuted(uint id);

Modifiers

GovernorAlpha has no modifiers.

Fields

GovernorAlpha contract has following fields and constants:

- string public constant name = "KeyFi Governance" – name of the contract
- TimelockInterface public timelock – the timelock address.
- TokenInterface public keyfi – the KeyFi token address.
- address public guardian – a guardian address.
- uint public proposalCount – the total number of proposals
- mapping (uint => Proposal) public proposals – all proposals.
- mapping (address => uint) public latestProposalIds – proposals of an address.
- bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
- bytes32 public constant BALLOT_TYPEHASH = keccak256("Ballot(uint256 proposalId,bool support)");

Functions

GovernorAlpha has following public functions:

- ***constructor***
  Description
  Inits the contract and sets default parameters.
  Visibility
  public
  Input parameters

- address timelock_ - the time contract address.
- address keyfi_ - the KeyFi token address.
- address guardian_ - a guardian address.

Constraints
None
Events emit
None
Output
None

- ***propose***

Description
Propose a new vote.
Visibility
public
Input parameters

- address[] memory targets – transaction targets.
- uint[] memory values – transaction values.
- string[] memory signatures – functions to be executed.
- bytes[] memory calldatas – calldata of calls.
- string memory description – a propose description.

Constraints

- A proposer votes should be above the proposal threshold.
- `targets`, `values`, `signatures`, `calldatas` should be of the same length.
- `targets` length should be more than 0.
- `targets` length should be less than proposalMaxOperations.
- A proposer can have one live.

Events emit
Emits the `ProposalCreated` event.
Output

- uint – a propose id.

- ***queue***

Description
Queue a proposal for the execution.
Visibility
public
Input parameters

- uint proposalId – a proposal id.

Constraints

- A proposal can only be queued only if it is succeeded.

- A proposal should not be queued yet.

Events emit

Emits the `ProposalQueued` event.

Output

None

- ***execute***

Description

Execute a proposal.

Visibility

public

Input parameters

- uint proposalId – a proposal id.

Constraints

- A proposal should be queued.
- A proposal should not be executed yet.

Events emit

Emits the `ProposalExecuted` event.

Output

None

- ***cancel***

Description

Cancel a proposal.

Visibility

public

Input parameters

- uint proposalId – a proposal id.

Constraints

- A proposal should not be executed yet.
- A message sender should be a guardian or a proposer votes become less than threshold.

Events emit

Emits the `ProposalCanceled` event.

Output

None

- ***getActions***

Description

Get actions of a specified proposal.

Visibility

public view

Input parameters

- uint proposalId – a proposal id.

Constraints

None

Events emit

None

Output

- address[] memory targets
- uint[] memory values
- string[] memory signatures
- bytes[] memory calldatas

- ***getReceipt***

  Description

  Get a vote results of a `voter` in a provided proposal.

  Visibility

  public view

  Input parameters

  - uint proposalId – a proposal id.
  - address voter – a voter address.

  Constraints

  None

  Events emit

  None

  Output

  - Receipt memory

- ***state***

  Description

  Get a state of a given proposal.

  Visibility

  public view

  Input parameters

  - uint proposalId – a proposal id.

  Constraints

  None

  Events emit

  None

  Output

  - ProposalState

- ***state***

  Description

  Get a state of a given proposal.

Visibility
public view
Input parameters
- uint proposalId – a proposal id.

Constraints
None
Events emit
None
Output
- ProposalState

- ***castVote***
Description
Participate in a vote.
Visibility
public
Input parameters
- uint proposalId – a proposal id.
- bool support

Constraints
- A proposal should be active.
- A voter should not participate yet.

Events emit
Emits the `VoteCast` event
Output
None

- ***castVoteBySig***
Description
Participate in a vote on behalf of another voter.
Visibility
public
Input parameters
- uint proposalId – a proposal id.
- bool support
- uint8 v – part of a signature.
- bytes32 r – part of a signature.
- bytes32 s – part of a signature.

Constraints
- A proposal should be active.
- A voter should not participate yet.

Events emit

Emits the `VoteCast` event
Output
None

- ***__acceptAdmin***
  Description
  Accept admin rights of the Timelock contract.
  Visibility
  public
  Input parameters
  None
  Constraints
  - A message sender should be a guardian.

  Events emit
  None
  Output
  None
- ***__abdicate***
  Description
  Removes a guardian address.
  Visibility
  public
  Input parameters
  None
  Constraints
  - A message sender should be a guardian.

  Events emit
  None
  Output
  None
- ***__queueSetTimelockPendingAdmin***
  Description
  Queue a new pending admin of the Timelock contract.
  Visibility
  public
  Input parameters
  - address newPendingAdmin – an address of the new admin.
  - uint eta – execution time.

  Constraints
  - A message sender should be a guardian.

  Events emit

None
Output
None

- ***executeSetTimelockPendingAdmin***
  Description
  Set a new pending admin of the Timelock contract.
  Visibility
  public
  Input parameters
  - address newPendingAdmin – an address of the new admin.
  - uint eta – execution time.

  Constraints
  - A message sender should be a guardian.
  - A transaction should be previously queued.

  Events emit
  None
  Output
  None

Timelock.sol

Description

Timelock queues and executes transactions.

Imports

Timelock has following imports:

- SafeMath.sol – from the OpenZeppelin.

Inheritance

Timelock does not inherit anything.

Usages

Timelock contract has following usages:

- SafeMath for uint.

Structs

Timelock contract has no data structures.

Enums

Timelock contract has no enums.

Events

Timelock contract has following events:

- event NewAdmin(address indexed newAdmin);
- event NewPendingAdmin(address indexed newPendingAdmin);
- event NewDelay(uint indexed newDelay);
- event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
- event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
- event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

Modifiers

Timelock has no modifiers.

Fields

Timelock contract has following fields and constants:

- uint public constant GRACE_PERIOD = 14 days;
- uint public constant MINIMUM_DELAY = 2 days;
- uint public constant MAXIMUM_DELAY = 30 days;
- address public admin – an admin address.
- address public pendingAdmin – a pending adming.
- uint public delay – delay between a transaction queueing and execution.
- mapping (bytes32 => bool) public queuedTransactions – queued transactions.

Functions

Timelock has following public functions:

- ***constructor***
  Description
  Inits the contract and sets default parameters.
  Visibility
  public
  Input parameters

- address admin_ - a contract admin.
- uint delay_ - delay between a transaction queuing and execution.

Constraints

- A `delay_` value should be between DELAY and MAXIMUM_DELAY.

Events emit

None

Output

None

- ***receive***

  Description

  Allows to receive ETH.

- ***setDelay***

  Description

  Sets a delay.

  Visibility

  public

  Input parameters

  - uint delay_ - delay between a transaction queuing and execution.

  Constraints

  - A message sender should be the contract itself.
  - A `delay_` value should be between DELAY and MAXIMUM_DELAY.

  Events emit

  Emits the `NewDelay` event.

  Output

  None

- ***acceptAdmin***

  Description

  Accept the admin permissions.

  Visibility

  public

  Input parameters

  None

  Constraints

  - A message sender should be a pending admin.

  Events emit

  Emits the `NewAdmin` event.

  Output

  None

- ***setPendingAdmin***

  Description

Accept the admin permissions.
Visibility
public
Input parameters
- address pendingAdmin_ - a pending admin address.

Constraints
- A message sender should be the contract itself.

Events emit
Emits the `NewPendingAdmin` event.
Output
None

- ***queueTransaction***

Description
Add a new transaction to the queue.
Visibility
public
Input parameters
- address target – a tx target.
- uint value – a tx value.
- string memory signature – a method signature.
- bytes memory data – a tx data.
- uint eta – a minimum delay between a tx queuing and execution.

Constraints
- A message sender should be admin.
- `eta` should be more than current time plus delay value.

Events emit
Emits the `QueueTransaction` event.
Output
bytes32 – a tx hash.

- ***cancelTransaction***

Description
Cancel a transaction.
Visibility
public
Input parameters
- address target – a tx target.
- uint value – a tx value.
- string memory signature – a method signature.

- bytes memory data – a tx data.
- uint eta – a minimum delay between a tx queuing and execution.

Constraints

- A message sender should be admin.

Events emit

Emits the `CancelTransaction` event.

Output

None

- ***cancelTransaction***

Description

Execute a transaction.

Visibility

public

Input parameters

- address target – a tx target.
- uint value – a tx value.
- string memory signature – a method signature.
- bytes memory data – a tx data.
- uint eta – a minimum delay between a tx queuing and execution.

Constraints

- A message sender should be admin.
- A transaction should be queued.
- Current timestamp should be between `eta` and `eta` + GRACE_PERIOD.

Events emit

Emits the `ExecuteTransaction` event.

Output

None

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

Audit overview

Critical

1. The community-controlled governance contract will have permissions to mint tokens. As soon as its code is not provided, we consider it insecure to transfer tokens ownership to this contract.

   Fixed before the second audit.

2. The `removeStakingToken` function of the `RewardPool` contract can remove a token from the tokens list. This can lead to a situation when a user will not be able to withdraw his tokens.

   Fixed before the second audit.

High

1. The `adminWithdrawReward` function of the `RewardPool` allows an owner to transfer any amount of reward (KeyFi) tokens from the rewards pool without any informing of the community.

   Fixed before the second audit.

Medium

1. None of the contracts except the `RewardPool` has unit tests.
2. The `quorumVotes` votes values of the GovernorAlpha is too low. It's possible to create and execute a transaction if a holder has at least 400000 KeyFi tokens. As soon as KeyFi team will have a majority of votes, they will dictate any actions that can be performed by the GovernonAlpha contract.
3. The `approve` function does not have a `spender` address validation.

   Fixed before the second audit.

4. *`delegateBySig`*, *`getPriorVotes`* and *`delegate`* functions of the KeyfiToken are never called internally so their type can be changed to external to reduce gas consumption.

   Fixed before the second audit.

5. Old compiler version is used. We recommend updating to the latest stable version.

Fixed before the second audit.

Low

1. Custom SafeMath function are used in the `KeyfiToken` contract. They can be replaced with the OpenZeppeling SafeMath functions to reduce code duplication.
2. `quorumVotes`, `proposalThreshold`, `proposalMaxOperations`, `votingDelay`, `votingPeriod` functions can be replaced with the corresponding constants.

Lowest / Code style / Best Practice

1. The `KeyfiToken` contract contains commented out events. They can be removed to clean up the code.

   Fixed before the second audit.

2. The `execute` function of the `GovernorAlpha` contract contains commented out code.

   Fixed before the second audit.

3. The `Timelock` contract name does not match its real purpose. Such name is traditionally used for contracts that are responsible for funds storage.

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.

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.

As a result of the **first audit**, security engineers found 1 critical, 3 high and 4 medium severity issues during the audit.

After the **second audit**, the contract contains 2 medium, 2 low and 1 lowest severity issues.

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

Category	Check Item	Comments
Code review	▪ Repository consistency	▪ No unit tests exist for all contracts except the RewardPool.
	▪ Style guide violation	▪ Some contracts contain commented out code.

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.