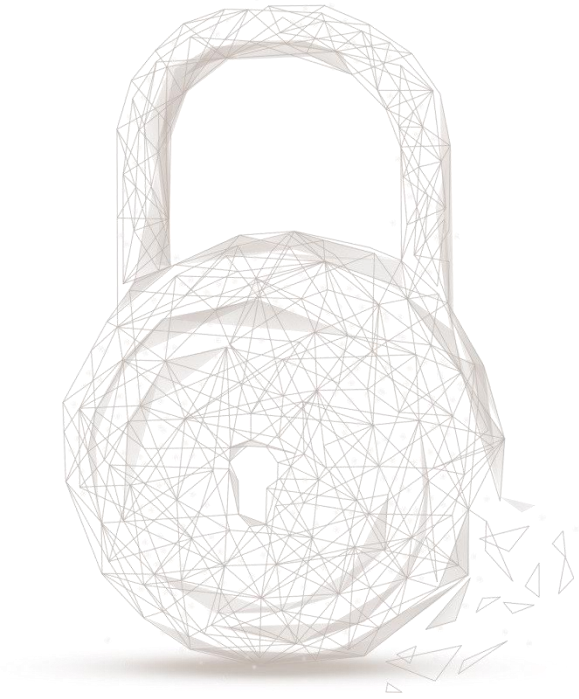
成都链安
BEOSIN

Smart contract security audit report



Audit Number：202010101010

Smart Contract Name：

CNHC Token (CNHC)

Smart Contract Address：

None

Smart Contract Address Link：

None

Start Date：2020.09.21

Completion Date：2020.10.10

Overall Result：Pass（Distinction）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass

3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	Missing
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract CNHC, including Coding Standards, Security, and Business Logic. **CNHC contract passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、Basic Token Information

Token name	CNHC Token
Token symbol	CNHC
decimals	18 (Can be modified when deploy)
totalSupply	Not deployed (Mintable without cap; Destroyable)
Token type	ERC20

Table 1 - Basic Token Information

2、Token Vesting Information

Missing

3、Custom Function Audit

- Transfer with fee

The contract implements the *setFeeParams* function. The contract owner can call this function to set the transfer fee-related parameters; after the transfer fee-related parameters are set, according to the amount and related transfer parameter, the internal function *calcFee* is called to calculate the transfer fee when transferring tokens, the fee is sent to the address of the contract owner, and the remaining part is normally transferred to the token receiving address.

- Blacklist management

The contract implements the blacklist management function. The contract owner can call related functions to add or remove the specified address from the blacklist. The address added to the blacklist cannot participate in token transfer, transferFrom, and approve(increase/decrease allowance) functions.

In particular, **the contract owner can destroy all CNHC token assets of the blacklist address when the current contract is not upgraded**, and has communicated with the project party during the audit period, ensuring that this function is the authority required by the project party's token function, and the contract owner authority audit is pass, special descriptions are given here.

- Pausable contract

The contract implements the Pausable module related functions. The contract owner can call related functions to pause or unpause the contract transfer status. When the contract is paused, token transfer, transferFrom, token minting and token destruction cannot be performed.

- Upgradable contract

The contract implements the upgrade function. The contract owner can call the *deprecate* function to set the upgrade contract address. After the contract is upgraded, the main functions of the contract (transfer/transferFrom/approve/increaseAllowance/decreaseAllowance/balanceOf/totalSupply)perform related operations in the new contract through the specified function interfaces, and the pause module, minting, and destruction functions in the original contract were deprecated.

- Proposal vote governance

The contract implements the voting governance function. The contract owner can call the openAddVoterProposal, openRemoveVoterProposal, openMintProposal and openDestroyBlackFundsProposal

functions to create related proposals for voting governance. When the number of votes for a specified proposal exceeds half of the current total number of voters, the corresponding proposal will be executed.

In extreme cases, when the number of votes changes, the failed proposal becomes a passed issue. For example, when there are 5 voters in the contract, a minting proposal has been voted twice (not passed at this time), and when the "remove voter" proposal was voted to make the voter become 2 (votes have already been voted), the minting proposal that should have been passed at this time could only be closed by the owner address because the vote could not be confirmed.

In addition, the contract owner can control to close any proposal.

Audited Source Code with Comments:

```
// Beosin (Chengdu LianAn) // File: BlackList.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import "@openzeppelin/contracts/access/Ownable.sol";

contract BlackList is Ownable{

    mapping (address => bool) public blackList; // Beosin (Chengdu LianAn) // Declare the mapping variable 'blackList' for storing status of whether the specified address is on the blacklist.

    event AddedBlackList(address _user); // Beosin (Chengdu LianAn) // Declare the event 'AddedBlackList'.

    event RemovedBlackList(address _user); // Beosin (Chengdu LianAn) // Declare the event 'RemovedBlackList'.

    constructor() internal{}
    // Beosin (Chengdu LianAn) // Function 'addBlackList' for owner adding the specified address '_user' into the blacklist.
    function addBlackList(address _user) external onlyOwner {
        blackList[_user] = true;
        emit AddedBlackList(_user); // Beosin (Chengdu LianAn) // Trigger the event 'AddedBlackList'.
    }
    // Beosin (Chengdu LianAn) // Function 'removeBlackList' for owner removing the specified address '_user' from the blacklist.
    function removeBlackList(address _user) external onlyOwner {
        blackList[_user] = false;
        emit RemovedBlackList(_user); // Beosin (Chengdu LianAn) // Trigger the event 'RemovedBlackList'.
    }
    // Beosin (Chengdu LianAn) // The function 'isBlackListUser' is defined to query the blacklist status of specified address.
    function isBlackListUser(address _user) public view returns (bool){
        return blackList[_user];
    }
    // Beosin (Chengdu LianAn) // Modifier, require that the specified '_user' is not in the blacklist.
    modifier isNotBlackUser(address _user) {
```

```
        require(!isBlackListUser(_user), "BlackList: this address is in blacklist");
        _;
    }

}


// Beosin (Chengdu LianAn) // File: ERC20.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import "@openzeppelin/contracts/GSN/Context.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.
    using Address for address;

    mapping (address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_balances' for storing the token balance of corresponding address.
```

```
    mapping (address => mapping (address => uint256)) private _allowances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two addresses.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing the total token supply.

    string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the token name.
    string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the token symbol.
    uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing the token decimals.

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name, string memory symbol) public {
        _name = name; // Beosin (Chengdu LianAn) // Initialize the token name.
        _symbol = symbol; // Beosin (Chengdu LianAn) // Initialize the token symbol.
        _decimals = 18; // Beosin (Chengdu LianAn) // Initialize the token decimals to 18 as default.
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
```

```
    * Tokens usually opt for a value of 18, imitating the relationship between
    * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
    * called.
    *
    * NOTE: This information is only used for _display_ purposes: it in
    * no way affects any of the arithmetic of the contract, including
    * {IERC20-balanceOf} and {IERC20-transfer}.
    */
    function decimals() public view returns (uint8) {
        return _decimals;
    }

    /**
    * @dev See {IERC20-totalSupply}.
    */
    function totalSupply() public virtual view override returns (uint256) {
        return _totalSupply;
    }

    /**
    * @dev See {IERC20-balanceOf}.
    */
    function balanceOf(address account) public virtual view override returns (uint256) {
        return _balances[account];
    }

    /**
    * @dev See {IERC20-transfer}.
    *
    * Requirements:
    *
    * - `recipient` cannot be the zero address.
    * - the caller must have a balance of at least `amount`.
    */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        return true;
    }

    /**
    * @dev See {IERC20-allowance}.
    */
    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
```

```
    * @dev See {IERC20-approve}.
    *
    * Requirements:
    *
    * - `spender` cannot be the zero address.
    */
   // Beosin (Chengdu LianAn) // Beware that changing an allowance with this method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering.
   // Beosin (Chengdu LianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter allowance is recommended.
   function approve(address spender, uint256 amount) public virtual override returns (bool) {
       _approve(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to set the allowance which 'msg.sender' allowed to 'spender'.
       return true;
   }

   /**
    * @dev See {IERC20-transferFrom}.
    *
    * Emits an {Approval} event indicating the updated allowance. This is not
    * required by the EIP. See the note at the beginning of {ERC20};
    *
    * Requirements:
    * - `sender` and `recipient` cannot be the zero address.
    * - `sender` must have a balance of at least `amount`.
    * - the caller must have allowance for ``sender``'s tokens of at least
    * `amount`.
    */
   function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
       _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
       _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
       return true;
   }

   /**
    * @dev Atomically increases the allowance granted to `spender` by the caller.
    *
    * This is an alternative to {approve} that can be used as a mitigation for
    * problems described in {IERC20-approve}.
    *
    * Emits an {Approval} event indicating the updated allowance.
    *
    * Requirements:
    *
    * - `spender` cannot be the zero address.
```

```
    */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to increase the allowance between two addresses.
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to decrease the allowance between two addresses.
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'sender'.
        require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'recipient'. Avoid losing transferred tokens.
```

```
        _beforeTokenTransfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the function '_beforeTokenTransfer' to check the corresponding conditions before token transfer.

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'sender'.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'recipient'.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _beforeTokenTransfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Call the function '_beforeTokenTransfer' to check the corresponding conditions before token minting.

        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        _balances[account] = _balances[account].add(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _beforeTokenTransfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Call the function
```

'_beforeTokenTransfer' to check the corresponding conditions before token destruction.

```
        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'owner'.
        require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.

        _allowances[owner][spender] = amount; // Beosin (Chengdu LianAn) // The allowance which 'owner' allowed to 'spender' is set to 'amount'.
        emit Approval(owner, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
```

```
    *
    * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
    * will be to transferred to `to`.
    * - when `from` is zero, `amount` tokens will be minted for `to`.
    * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
    * - `from` and `to` are never both zero.
    *
    * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
    */
   function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}


// Beosin (Chengdu LianAn) // File: ERC20WithFee.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import "@openzeppelin/contracts/access/Ownable.sol";
import "./ERC20.sol";

contract ERC20WithFee is ERC20, Ownable {

  uint256 public basisPointsRate = 0; // Beosin (Chengdu LianAn) // Declare the variables 'basisPointsRate' for storing the basic fee rate, it is 0 as default.
  uint256 public maximumFee = 0; // Beosin (Chengdu LianAn) // Declare the variables 'maximumFee' for storing the maximum fee amount, it is 0 as default.
  // Beosin (Chengdu LianAn) // The constructor of contract 'ERC20WithFee'.
  constructor (string memory name, string memory symbol) public ERC20(name, symbol) {}
  // Beosin (Chengdu LianAn) // The internal function 'calcFee' is defined to calculate the fee amount.
  function calcFee(uint _value) internal view returns (uint) {
    uint fee = (_value.mul(basisPointsRate)).div(10000); // Beosin (Chengdu LianAn) // Calculate the initial fee amount of this transfer transaction based on the 'basisPointsRate' and '_value'.
    if (fee > maximumFee) {
        fee = maximumFee; // Beosin (Chengdu LianAn) // Change the transfer fee to 'maximumFee' if the current fee is higher than the 'maximumFee'.
    }
    return fee;
  }
  // Beosin (Chengdu LianAn) // The 'transfer' function, 'msg.sender' transfers the specified amount of tokens to a specified address.
  function transfer(address _to, uint _value) public override virtual returns (bool) {
    uint fee = calcFee(_value);
    uint sendAmount = _value.sub(fee); // Beosin (Chengdu LianAn) // Declare the local variable 'sendAmount' for calculating the actual sending amount of this transfer transaction.

    super.transfer(_to, sendAmount); // Beosin (Chengdu LianAn) // Call the transfer function in the parent contract to transfer the actual amount.
    if (fee > 0) {
```

```
        super.transfer(owner(), fee); // Beosin (Chengdu LianAn) // Call the transfer function in the parent contract to transfer the handling fee to the contract owner.
    }
  }
  // Beosin (Chengdu LianAn) // The 'transferFrom' function, 'msg.sender' as a delegate of '_from' to transfer the specified amount of tokens to a specified address.
  function transferFrom(address _from, address _to, uint256 _value) public override virtual returns (bool) {
    require(_to != address(0), "ERC20WithFee: transfer to the zero address");
    require(_value <= balanceOf(_from), "ERC20WithFee: transfer amount exceeds balance");
    require(_value <= allowance(_from, msg.sender), "ERC20WithFee: allowance amount exceeds allowed");

    uint fee = calcFee(_value);
    uint sendAmount = _value.sub(fee); // Beosin (Chengdu LianAn) // Declare the local variable 'sendAmount' for calculating the actual sending amount of this transfer transaction.

    _transfer(_from, _to, sendAmount); // Beosin (Chengdu LianAn) // Call the '_transfer' function to transfer the actual amount.
    if (fee > 0) {
        _transfer(_from, owner(), fee); // Beosin (Chengdu LianAn) // Call the '_transfer' function to transfer the handling fee to the contract owner.
    }
    _approve(_from, msg.sender, allowance(_from, msg.sender).sub(_value, "ERC20WithFee: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
    return true;
  }
  // Beosin (Chengdu LianAn) // Function 'setFeeParams' for owner setting the transfer relevant fee rate parameters.
  function setFeeParams(uint256 newBasisPoints, uint256 newMaxFee) public onlyOwner {
      basisPointsRate = newBasisPoints;
      maximumFee = newMaxFee.mul(uint256(10)**decimals());
      emit FeeParams(basisPointsRate, maximumFee); // Beosin (Chengdu LianAn) // Trigger the event 'FeeParams'.
  }

  // Called if contract ever adds fees
  event FeeParams(uint feeBasisPoints, uint maxFee); // Beosin (Chengdu LianAn) // Declare the event 'FeeParams'

}


// Beosin (Chengdu LianAn) // File: UpgradedStandardToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import "./ERC20WithFee.sol";
// Beosin (Chengdu LianAn) // The 'UpgradedStandardToken' contract declares the interfaces related to
```

```
contract upgrade.
abstract contract UpgradedStandardToken is ERC20WithFee {
    // those methods are called by the legacy contract
    // and they must ensure msg.sender to be the contract address
    uint public _totalSupply;
    function transferByLegacy(address from, address to, uint value) public virtual returns (bool);
    function transferFromByLegacy(address sender, address from, address spender, uint value) public virtual returns
(bool);
    function approveByLegacy(address from, address spender, uint value) public virtual returns (bool);
    function increaseApprovalByLegacy(address from, address spender, uint addedValue) public virtual returns
(bool);
    function decreaseApprovalByLegacy(address from, address spender, uint subtractedValue) public virtual returns
(bool);
}


// Beosin (Chengdu LianAn) // File: Votable.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import "@openzeppelin/contracts/access/Ownable.sol";

contract Votable is Ownable{

    // voter->bool
    mapping(address => bool) public voters; // Beosin (Chengdu LianAn) // Declare the mapping variable
'voters' for storing the voter permission status.

    // voters count
    uint16 public votersCount = 0; // Beosin (Chengdu LianAn) // Declare the variable 'votersCount' for storing
the voter counts.

    // pid->proposal
    mapping(uint16 => Proposal) public proposals; // Beosin (Chengdu LianAn) // Declare the variable
'proposals' for storing all proposal data.

    // next pid, start with 10000
    uint16 public nextPid = 10000;

    constructor() internal{
        // init owner as a voter
        voters[owner()] = true; // Beosin (Chengdu LianAn) // Initialize the contract owner to the voter.
        votersCount++;
        emit AddVoter(owner()); // Beosin (Chengdu LianAn) // Trigger the event 'AddVoter'.
    }

    // struct of proposal
    struct Proposal {
```

```
        uint16 pid;
        uint16 count;
        bool done;
        bytes payload;
        // voter->bool
        mapping(address => bool) votes;
    }

    // events
    event OpenProposal(uint16 pid);

    event CloseProposal(uint16 pid);

    event DoneProposal(uint16 pid);

    event VoteProposal(uint16 pid, address voter);

    event AddVoter(address voter);

    event RemoveVoter(address voter);

    // modifiers
    // Beosin (Chengdu LianAn) // Modifier, require that the corresponding proposal of the specified '_pid' should exist and has not been executed.
    modifier proposalExistAndNotDone(uint16 _pid){
        require(proposals[_pid].pid == _pid, "Votable: proposal not exists");
        require(!proposals[_pid].done, "Votable: proposal is done");
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, require that the caller of the modified function should be voter.
    modifier onlyVoters(){
        require(voters[msg.sender], "Votable: only voter can call");
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, require that the caller of the modified function should be this contract address.
    modifier onlySelf(){
        require(msg.sender == address(this), "Votable: only self can call");
        _;
    }

    // for inheriting
    // Beosin (Chengdu LianAn) // The internal function '_openProposal' is defined to create a proposal.
    function _openProposal(bytes memory payload) internal{
        uint16 pid = nextPid++; // Beosin (Chengdu LianAn) // Get the proposal id.
        proposals[pid] = Proposal(pid,0,false,payload); // Beosin (Chengdu LianAn) // Store the specified proposal data.
        emit OpenProposal(pid); // Beosin (Chengdu LianAn) // Trigger the event 'OpenProposal'.
```

```
    }

    // vote
    function voteProposal(uint16 _pid) public onlyVoters proposalExistAndNotDone(_pid){
        Proposal storage proposal = proposals[_pid]; // Beosin (Chengdu LianAn) // Get the specified proposal data.
        require(!proposal.votes[msg.sender], "Votable: duplicate voting is not allowed"); // Beosin (Chengdu LianAn) // Require that each vote can vote only once.
        // Beosin (Chengdu LianAn) // Update the vote counts and the vote status of caller.
        proposal.votes[msg.sender] = true;
        proposal.count++;
        emit VoteProposal(_pid, msg.sender); // Beosin (Chengdu LianAn) // Trigger the event 'VoteProposal'.

        // judge
        _judge(proposal); // Beosin (Chengdu LianAn) // Call the function '_judge' to judge whether the specified proposal passed.
    }

    function _judge(Proposal storage _proposal) private{
        if(_proposal.count > votersCount/2){ // Beosin (Chengdu LianAn) // If the vote count exceeds the half of total voter number, execute the specified proposal via call operation.
            (bool success, ) = address(this).call(_proposal.payload);
            require(success, "Votable: call payload failed"); // Beosin (Chengdu LianAn) // Check the proposal execution result.
            _proposal.done = true;
            emit DoneProposal(_proposal.pid); // Beosin (Chengdu LianAn) // Trigger the event 'DoneProposal'.
        }
    }

    // hasVoted
    function hasVoted(uint16 _pid) public view returns(bool){
        Proposal storage proposal = proposals[_pid];
        require(proposal.pid == _pid, "Votable: proposal not exists");
        return proposal.votes[msg.sender];
    }

    // translate proposal
    // function translateProposal(uint16 _pid) external view returns(bytes32, address, uint256){
    //     Proposal memory proposal = proposals[_pid];
    //     require(proposal.pid == _pid, "Votable: proposal not exists");
    //     return abi.decode(abi.encodePacked(bytes28(0), proposal.payload),(bytes32,address,uint256));
    // }

    // onlySelf: match to proposals
    // Beosin (Chengdu LianAn) // The function 'addVoter' is defined to add voter.
    function addVoter(address _voter) external onlySelf{
        require(!voters[_voter], "Votable: this address is already a voter"); // Beosin (Chengdu LianAn) // Require that the same voter cannot be added repeatedly.
```

```
        // Beosin (Chengdu LianAn) // Update the voter count and the corresponding voter permission of '_voter'.
        voters[_voter] = true;
        votersCount++;
        emit AddVoter(_voter); // Beosin (Chengdu LianAn) // Trigger the event 'AddVoter'.
    }
    // Beosin (Chengdu LianAn) // The function 'removeVoter' is defined to remove voter.
    function removeVoter(address _voter) external onlySelf{
        require(voters[_voter], "Votable: this address is not a voter"); // Beosin (Chengdu LianAn) // Require that the voter to be removed should exist.
        require(_voter != owner(), "Votable: owner can not be removed"); // Beosin (Chengdu LianAn) // Require that the voter permission of the owner cannot be removed.
        // Beosin (Chengdu LianAn) // Update the voter count and the corresponding voter permission of '_voter'.
        voters[_voter] = false;
        votersCount--;
        emit RemoveVoter(_voter); // Beosin (Chengdu LianAn) // Trigger the event 'RemoveVoter'.
    }

    // onlyOwner
    // open proposals
    function openAddVoterProposal(address _voter) external onlyOwner{
        _openProposal(abi.encodeWithSignature("addVoter(address)",_voter)); // Beosin (Chengdu LianAn) // Call the function '_openProposal' to create a specified proposal to add a voter.
    }

    function openRemoveVoterProposal(address _voter) external onlyOwner{
        _openProposal(abi.encodeWithSignature("removeVoter(address)",_voter)); // Beosin (Chengdu LianAn) // Call the function '_openProposal' to create a specified proposal to remove a voter.
    }

    // close proposal
    function closeProposal(uint16 _pid) external proposalExistAndNotDone(_pid) onlyOwner{
        proposals[_pid].done = true; // Beosin (Chengdu LianAn) // Change the execution status of specified proposal to true(close the specified proposal), making it cannot be voted for execution.
        emit CloseProposal(_pid); // Beosin (Chengdu LianAn) // Trigger the event 'CloseProposal'.
    }

}


// Beosin (Chengdu LianAn) // File: CNHCToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/GSN/Context.sol";
import "@openzeppelin/contracts/utils/Pausable.sol";
```

```
import "./BlackList.sol";
import "./Votable.sol";
import "./UpgradedStandardToken.sol";
import "./ERC20WithFee.sol";

contract CNHCToken is ERC20WithFee, BlackList, Votable, Pausable {

    address public upgradedAddress; // Beosin (Chengdu LianAn) // Declare the variable 'upgradedAddress' for storing the new token address after upgrading.

    bool public deprecated; // Beosin (Chengdu LianAn) // Declare the variable 'deprecated' for storing the deprecate status of this contract.

    constructor(uint256 _initialSupply, uint8 _decimals) public ERC20WithFee("CNHC Token","CNHC") {
        _setupDecimals(_decimals); // Beosin (Chengdu LianAn) // Call the internal function '_setupDecimals' to set the token decimals. Note: the token decimals are initialized to 18 in the constructor of ERC20 contract.
        _mint(_msgSender(), _initialSupply); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to mint tokens to the deployer.
    }

    event DestroyedBlackFunds(address indexed blackListedUser, uint balance); // Beosin (Chengdu LianAn) // Declare the event 'DestroyedBlackFunds'.

    event Deprecate(address newAddress); // Beosin (Chengdu LianAn) // Declare the event 'Deprecate'.

    // functions users can call
    // make compatible if deprecated
    // Beosin (Chengdu LianAn) // Rewrite the function 'balanceOf'.
    function balanceOf(address account) public override view returns (uint256) {
        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return UpgradedStandardToken(upgradedAddress).balanceOf(account); // Beosin (Chengdu LianAn) // Call the function 'balanceOf' of contract 'UpgradedStandardToken' to query the actual token balance in the new contract.
        } else {
            return super.balanceOf(account); // Beosin (Chengdu LianAn) // Return the token balance in the current contract.
        }
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'totalSupply', added the logic of returning the actual totalSupply in the upgraded contract.
    function totalSupply() public override view returns (uint) {
        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return IERC20(upgradedAddress).totalSupply(); // Beosin (Chengdu LianAn) // Call the function 'totalSupply' of the upgraded contract to query the total token supply.
        } else {
            return super.totalSupply(); // Beosin (Chengdu LianAn) // Return the total token supply of the current
```

```
contract.
        }
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'allowance'.
    function allowance(address owner, address spender) public override view returns (uint remaining) {
        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return IERC20(upgradedAddress).allowance(owner, spender); // Beosin (Chengdu LianAn) // Call the function 'allowance' of the upgraded contract to query the actual allowance between specified address 'owner' and 'spender'.
        } else {
            return super.allowance(owner, spender); // Beosin (Chengdu LianAn) // Return the allowance between specified address 'owner' and 'spender' in the current contract.
        }
    }

    // Allow checks of balance at time of deprecation
    function oldBalanceOf(address account) public view returns (uint256) {
        if (deprecated) {
            return super.balanceOf(account); // Beosin (Chengdu LianAn) // Return the total token supply of the old contract when the contract is deprecated.
        }
    }

    // normal functions
    // Beosin (Chengdu LianAn) // Rewrite the function 'transfer', added the logic of checking blacklist and deprecated executing.
    function transfer(address recipient, uint256 amount) public override isNotBlackUser(_msgSender()) returns (bool) {
        require(!isBlackListUser(recipient), "BlackList: recipient address is in blacklist"); // Beosin (Chengdu LianAn) // Blacklist check, require that the specified address 'recipient' is not in the blacklist.
        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return UpgradedStandardToken(upgradedAddress).transferByLegacy(_msgSender(), recipient, amount); // Beosin (Chengdu LianAn) // Call the function 'transferByLegacy' of contract 'UpgradedStandardToken' to do the corresponding operation.
        } else {
            return super.transfer(recipient, amount); // Beosin (Chengdu LianAn) // Execute the function 'transfer' of the parent contract and return the result.
        }
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'transferFrom', added the logic of checking blacklist and deprecated executing.
    function transferFrom(address sender, address recipient, uint256 amount) public override isNotBlackUser(_msgSender()) returns (bool) {
        require(!isBlackListUser(sender), "BlackList: sender address is in blacklist"); // Beosin (Chengdu LianAn) // Blacklist check, require that the specified address 'sender' is not in the blacklist.
        require(!isBlackListUser(recipient), "BlackList: recipient address is in blacklist"); // Beosin (Chengdu
```

```
LianAn) // Blacklist check, require that the specified address 'recipient' is not in the blacklist.

        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return UpgradedStandardToken(upgradedAddress).transferFromByLegacy(_msgSender(), sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the function 'transferFromByLegacy' of contract 'UpgradedStandardToken' to do the corresponding operation.
        } else {
            return super.transferFrom(sender, recipient, amount); // Beosin (Chengdu LianAn) // Execute the function 'transferFrom' of the parent contract and return the result.
        }
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'approve', added the logic of checking blacklist and deprecated executing.
    function approve(address spender, uint256 amount) public override isNotBlackUser(_msgSender()) returns (bool) {
        require(!isBlackListUser(spender), "BlackList: spender address is in blacklist"); // Beosin (Chengdu LianAn) // Blacklist check, require that the specified address 'spender' is not in the blacklist.
        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return UpgradedStandardToken(upgradedAddress).approveByLegacy(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call the function 'approveByLegacy' of contract 'UpgradedStandardToken' to do the corresponding operation.
        } else {
            return super.approve(spender, amount); // Beosin (Chengdu LianAn) // Execute the function 'approve' of the parent contract and return the result.
        }
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'increaseAllowance', added the logic of checking blacklist and deprecated executing.
    function increaseAllowance(address spender, uint256 addedValue) public override isNotBlackUser(_msgSender()) returns (bool) {
        require(!isBlackListUser(spender), "BlackList: spender address is in blacklist"); // Beosin (Chengdu LianAn) // Blacklist check, require that the specified address 'spender' is not in the blacklist.

        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return UpgradedStandardToken(upgradedAddress).increaseApprovalByLegacy(_msgSender(), spender, addedValue); // Beosin (Chengdu LianAn) // Call the function 'increaseApprovalByLegacy' of contract 'UpgradedStandardToken' to do the corresponding operation.
        } else {
            return super.increaseAllowance(spender, addedValue); // Beosin (Chengdu LianAn) // Execute the function 'increaseAllowance' of the parent contract and return the result.
        }
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'decreaseAllowance', added the logic of checking blacklist and deprecated executing.
    function decreaseAllowance(address spender, uint256 subtractedValue) public override
```

```
isNotBlackUser(_msgSender()) returns (bool) {
        require(!isBlackListUser(spender), "BlackList: spender address is in blacklist"); // Beosin (Chengdu LianAn) // Blacklist check, require that the specified address 'spender' is not in the blacklist.

        if (deprecated) { // Beosin (Chengdu LianAn) // According to the check result of whether this contract is upgraded, enter different branches.
            return UpgradedStandardToken(upgradedAddress).decreaseApprovalByLegacy(_msgSender(), spender, subtractedValue); // Beosin (Chengdu LianAn) // Call the function 'decreaseApprovalByLegacy' of contract 'UpgradedStandardToken' to do the corresponding operation.
        } else {
            return super.decreaseAllowance(spender, subtractedValue); // Beosin (Chengdu LianAn) // Execute the function 'decreaseAllowance' of the parent contract and return the result.
        }
    }
    // Beosin (Chengdu LianAn) // The 'burn' function, 'msg.sender' burns the specific amount of tokens.
    function burn(uint256 amount) public {
        require(!deprecated, "CNHCToken: contract was deprecated"); // Beosin (Chengdu LianAn) // Require that the current contract is not deprecated.
        super._burn(_msgSender(), amount); // Beosin (Chengdu LianAn) // Execute the function '_burn' of the parent contract to destroy tokens.
    }

    // functions only owner can call
    // open proposals
    function openMintProposal(address _account, uint256 _amount) external onlyOwner{
        _openProposal(abi.encodeWithSignature("mint(address,uint256)", _account, _amount)); // Beosin (Chengdu LianAn) // Call the function '_openProposal' to create a specified proposal to mint tokens to a specified address.
    }

    function openDestroyBlackFundsProposal(address _user) external onlyOwner{
        _openProposal(abi.encodeWithSignature("destroyBlackFunds(address)", _user)); // Beosin (Chengdu LianAn) // Call the function '_openProposal' to create a specified proposal to destroy funds of a blacklist address.
    }

    // onlySelf: mint & burn
    function mint(address _account, uint256 _amount) public onlySelf {
        require(!deprecated, "CNHCToken: contract was deprecated");
        super._mint(_account, _amount);
    }

    function destroyBlackFunds(address _user) public onlySelf {
        require(!deprecated, "CNHCToken: contract was deprecated"); // Beosin (Chengdu LianAn) // Require that the current contract is not deprecated.
        require(isBlackListUser(_user), "BlackList: only fund in blacklist address can be destroy"); // Beosin (Chengdu LianAn) // Blacklist check, require that the specified address 'spender' is in the blacklist. Avoid destroy the funds of ordinary address.
```

```
        uint dirtyFunds = balanceOf(_user); // Beosin (Chengdu LianAn) // Declare the local variable 'dirtyFunds' for recording the token balance of '_user' in the current contract.
        super._burn(_user, dirtyFunds); // Beosin (Chengdu LianAn) // Execute the function '_burn' of the parent contract to destroy tokens.
        emit DestroyedBlackFunds(_user, dirtyFunds); // Beosin (Chengdu LianAn) // Trigger the event 'DestroyedBlackFunds'.
    }

    // pause
    function pause() public onlyOwner {
        require(!deprecated, "CNHCToken: contract was deprecated"); // Beosin (Chengdu LianAn) // Require that the current contract is not deprecated.
        super._pause(); // Beosin (Chengdu LianAn) // Execute the function '_pause' of the parent contract to change the pause status to true.
    }

    function unpause() public onlyOwner {
        require(!deprecated, "CNHCToken: contract was deprecated"); // Beosin (Chengdu LianAn) // Require that the current contract is not deprecated.
        super._unpause(); // Beosin (Chengdu LianAn) // Execute the function '_unpause' of the parent contract to change the pause status to false.
    }

    // deprecate
    // Beosin (Chengdu LianAn) // Note: this function can only be called once, cautiously using it is recommended.
    function deprecate(address _upgradedAddress) public onlyOwner {
        require(!deprecated, "CNHCToken: contract was deprecated"); // Beosin (Chengdu LianAn) // Require that the current contract is not deprecated.
        require(_upgradedAddress != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for '_upgradedAddress'.
        deprecated = true;
        upgradedAddress = _upgradedAddress;
        emit Deprecate(_upgradedAddress); // Beosin (Chengdu LianAn) // Trigger the event 'Deprecate'.
    }

    // hook before _transfer()/_mint()/_burn()
    // Beosin (Chengdu LianAn) // Rewrite the internal function '_beforeTokenTransfer', check the pause status.
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        require(!paused(), "Pausable: token transfer while paused"); // Beosin (Chengdu LianAn) // Require that the contract has not been paused.
    }
}
```

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com