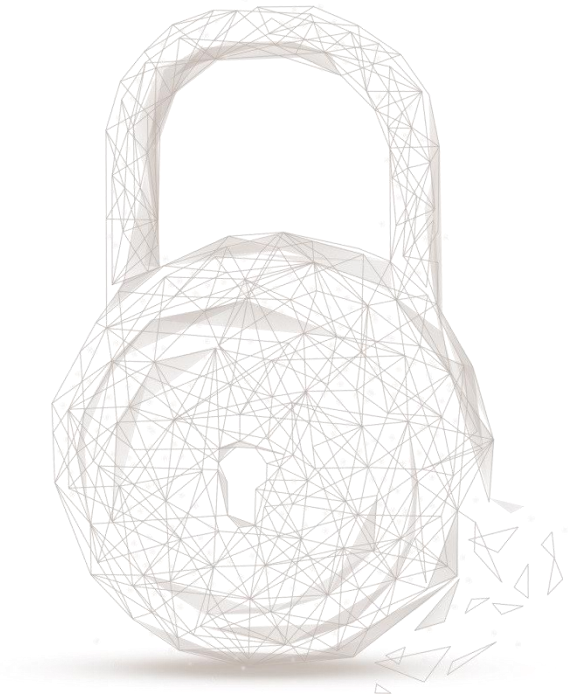
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202101041732

Smart Contract Name：

FC

Smart Contract Address：

TKzf6cF5RsxzV7Shcf8MpuRRJcfDiG1hDn

Smart Contract Address Link：

https://tronscan.org/#/contract/TKzf6cF5RsxzV7Shcf8MpuRRJcfDiG1hDn/code

Start Date：2020.01.04

Completion Date：2020.01.04

Overall Result：Pass（Distinction）

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	TRC-20 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	-	N/A
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

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 FC, including Coding Standards, Security, and Business Logic. **FC 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	Filecoin Community Token
Token symbol	FC
decimals	6
totalSupply	100 million
Token type	TRC-20

Table 1 – Basic Token Information

2、Token Vesting Information

N/A

3、Other function description

- transfer Function

As shown in the figure below, the contract modifies the transfer function of ERC20 in OpenZeppelin, and implements the logic of deducting 10% of the transfer fee.

```
231     function transfer(address recipient, uint256 amount) public returns (bool) {
232         require(amount >= 0, "Cannot transfer lower 0");
233         uint256 feeValue = SafeMath.div(amount, 10);
234         uint256 taxedValue = SafeMath.sub(amount, feeValue);
235         _transfer(msg.sender, recipient, taxedValue);
236         _transfer(msg.sender, feeAddr, feeValue);
237         return true;
238     }
```

Figure 1 transfer function source code

- internalTransfer Function

As shown in the figure below, the contract implements the internalTransfer function. Anyone can call this function to transfer money without handling fees.

```
450     function internalTransfer(address recipient, uint256 amount)
451         public
452         returns (bool)
453     {
454         _transfer(msg.sender, recipient, amount);
455         return true;
456     }
```

Figure 2 internalTransfer function source code

- setFeeAddr Function

As shown in the figure below, the contract implements the setFeeAddr function to modify the fee recipient address, and only the owner of the contract can call it.

```
461     function setFeeAddr(address _feeAddr) external onlyOwner returns (address) {
462         require(
463             _feeAddr != address(0),
464             "ERC20: _feeAddr set to the zero address"
465         );
466         feeAddr = _feeAddr;
467         return feeAddr;
468     }
```

Figure 3 setFeeAddr function source code

Audited Source Code with Comments:

```
// 0.5.1-c8a2
// Enable optimization
pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

// Beosin (Chengdu LianAn) // The SafeMath library declares functions for safe mathematical operation.
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
```

```
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
```

```
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}
// Beosin (Chengdu LianAn) // Define the function interfaces required by TRC-20 Token standard.
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
```

```
function totalSupply() external view returns (uint256);

/**
 * @dev Returns the amount of tokens owned by `account`.
 */
function balanceOf(address account) external view returns (uint256);

/**
 * @dev Moves `amount` tokens from the caller's account to `recipient`.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transfer(address recipient, uint256 amount) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: 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. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an {Approval} event.
```

```
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value); // Beosin (Chengdu LianAn) //
Declare the event 'Transfer'.

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value); // Beosin (Chengdu
LianAn) // Declare the event 'Approval'.
}
contract Ownable {
    address owner; // Beosin (Chengdu LianAn) // Declare the variable 'owner' to store the address of the
contract owner.
    address newOwner; // Beosin (Chengdu LianAn) // Declare the variable 'newOwner' to store the new
owner address of the contract.


    constructor() internal {
```

```
        owner = msg.sender; // Beosin (Chengdu LianAn) // Constructor, set the contract creator as 'owner'.

    // Beosin (Chengdu LianAn) // Modifier, check that the caller is 'owner'.
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
    // Beosin (Chengdu LianAn) // Change the 'newOwner' address, only 'owner' can call.
    function ChangeOwnership(address p_newOwner) external onlyOwner {
        newOwner = p_newOwner;
    }
    // Beosin (Chengdu LianAn) // The newOwner accepts ownership, only 'newOwner' can call.
    function AcceptOwnership() external {
        require(msg.sender == newOwner);
        owner = newOwner;
    }
    // Beosin (Chengdu LianAn) // Return the current 'owner' address.
    function ownner() public view returns (address) {
        return owner;
    }
}
contract ERC20 is IERC20, Ownable {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    address public feeAddr; // Beosin (Chengdu LianAn) // Declare the address variable 'feeAddr' to store the address for receiving the fee.

    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.

    constructor() internal {
```

```
        feeAddr = msg.sender; // Beosin (Chengdu LianAn) // Constructor, assign the deployer address to 'feeAddr' address variable.
    }

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

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view 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 returns (bool) {
        require(amount >= 0, "Cannot transfer lower 0"); // Beosin (Chengdu LianAn) // Check that the transfer amount is greater than or equal to 0.
        uint256 feeValue = SafeMath.div(amount, 10) // Beosin (Chengdu LianAn) // Calculate the transfer fee.
        uint256 taxedValue = SafeMath.sub(amount, feeValue); // Beosin (Chengdu LianAn) // Calculate the actual transfer amount.
        _transfer(msg.sender, recipient, taxedValue); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' for token transfer.
        _transfer(msg.sender, feeAddr, feeValue); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' for fee transfer.
        return true;
```

```
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender)
        public
        view
        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(ChengduLianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter Allowance is recommended.
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to set the caller's approval value for the sender.
        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 `value`.
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public returns (bool) {
        _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        _approve(
            sender,
            msg.sender,
            _allowances[sender][msg.sender].sub(amount) // 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
        returns (bool)
    {
        _approve(
            msg.sender,
```

```
            spender,
            _allowances[msg.sender][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
        returns (bool)
    {
        _approve(
            msg.sender,
            spender,
            _allowances[msg.sender][spender].sub(subtractedValue)
        ); // 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 {
        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.
        _balances[sender] = _balances[sender].sub(amount); // 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 {
        require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.
```

```
        _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.
     */
    // Beosin (Chengdu LianAn) // Redundant code, it is recommended to delete.
    function _burn(address account, uint256 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is 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 value
    ) internal {
        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] = value; // Beosin (Chengdu LianAn) // The allowance which 'owner' allowed to 'spender' is set to 'amount'.
        emit Approval(owner, spender, value); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
    }

    /**
     * @dev Destoys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    // Beosin (Chengdu LianAn) // Redundant code, it is recommended to delete.
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(
            account,
            msg.sender,
            _allowances[account][msg.sender].sub(amount)
        );
    }

    /**
     * @dev See {IERC20-transfer}.
     *
```

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

    /**
     * set fee
     */
    function setFeeAddr(address _feeAddr) external onlyOwner returns (address) {
        require(
            _feeAddr != address(0),
            "ERC20: _feeAddr set to the zero address"
        );
        feeAddr = _feeAddr; // Beosin (Chengdu LianAn) // Modify the value of the address variable 'feeAddr', only the owner can modify it.
        return feeAddr;
    }
}
contract ERC20Detailed is IERC20 {
    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`, `symbol`, and `decimals`. All three of
     * these values are immutable: they can only be set once during
     * construction.
```

```
 */
// Beosin(ChengduLianAn) // Constructor, initialize basic token information.
constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
}

/**
 * @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.
 *
 * 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;
}
```

```
}
contract FC is ERC20, ERC20Detailed {
    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    constructor() public ERC20Detailed("FC", "FC", 6) {
        _mint(msg.sender, 100000000 * (10**uint256(decimals()))); // Beosin (Chengdu LianAn) //
Constructor, minting 100 million tokens to the address that created the contract.
    }
}
// Beosin (Chengdu LianAn) // Recommend the main contract to inherit 'Pausable' module to grant owner
the authority of pausing all transactions when serious issue occurred.
```

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com