Dao Chain Technology

Smart Contract Security Audit Report

Supervised by Daolian Technology Company

Audit number： DX10220202258

Produced by Dao Chain Technology Co Ltd

Audit number:

DX10220202258

Token name:

Discrete cipher block(DCB)

Audit contract address:

0xE87E267E92c8f950fd194419c3538B7df6590909

Audit contract link address:

https://etherscan.io/address/0xE87E267E92c8f950fd194419c3538B7df6590909#code

Audit results:

After the audit, the (DCB) contract passed all tests, and the contract audit result was passed (excellent)

Smart contract permissions:

The basic contract does not have too many permissions

Audit team:

DAO CHAIN TECHNOLOGY COMPANY

Types of Audits and Results:

SN	Audit type	Audit subkey	Audit results
1	overflow audit	-	pass√

2	function call auditing	1. .call/delegatecall security audit 2. Function return value security audit 3. Self-destructing function security audit 4. Function call permission audit 5. Compiler version safety	pass√
3	Reentrant Attack Audit	-	pass√
4	Gas optimization audit	-	pass√
5	"Fake Top-Up" Vulnerability Audit	-	pass√
6	Pseudo-random number generation audit	-	pass√
7	Code Specification Audit	1. ERC-20 Token Standard Specification Audit 2. Redundant code audit 3. Variable coverage audit 4. Deprecated item auditing	pass√
8	Business Security Audit	1. owner permission audit 2. Business logic audit 3. Business implementation audit	pass√
9	Denial of Service Attack Audit	-	pass√

10	Block parameter dependency auditing	-	pass√
11	Abnormal reachability status audit	-	**pass**√

Note: Please refer to the code comments for audit opinions and suggestions!

(Dao Chain Technology Statement: Dao Chain Technology only issues this report based on the attacks or vulnerabilities that have existed or occurred before the issuance of this report. For new attacks or vulnerabilities that exist or occur after the issuance of this report, Dao Chain Technology cannot judge the security of smart contracts. We do not take any responsibility for the possible impact of the situation. The security audit analysis and other contents in this report are only based on the documents and information provided by the DCB contract provider to Dao Chain Technology before the issuance of this report, and this part of the documents and information Made under the premise that there is no defect, tampering, deletion or concealment; if the documents and materials provided have information lacking, tampering, deletion, concealment or the reflected situation does not match the actual situation or provide documents and information If there are any changes after the issuance of this report, Dao Chain Technology will not be responsible for any losses and adverse effects caused thereby.)

一.Basic information of Token

Token name	**Discrete cipher block**
Token symbol	**DCB**
decimals	**18**
totalSupply	**1,000,000,000**

Token type	**ERC-20**

The contract source code is as follows:

***//File: default project/ETH_.../comm.... sol**

1. Lack of zero address verification (481 lines)

2. Invalid code (110 lines)

3. Public functions can be declared as external functions (169 lines)

4. Public functions can be declared as external functions (177 lines)

5. Similar naming (470 lines)

6. Public functions can be declared as external functions (487 lines)

7. Public functions can be declared as external functions (495 lines)

8. Public functions can be declared as external functions (line 512)

9. Public functions can be declared as external functions (line 519)

10. Public functions can be declared as external functions (526 lines)

11. Public functions can be declared as external functions (544 lines)

12. Public functions can be declared as external functions (557 lines)

13. Public functions can be declared as external functions (line 574)

14. Public functions can be declared as external functions (line 597)

15. Public functions can be declared as external functions (626 lines)

16. Public functions can be declared as external functions (line 653)

17. Invalid code (731 lines)

18. Invalid code (776 lines)

pragma solidity ^0.8.4;// Dao Chain Technology // It is recommended to fix the compiler version

* (The 446 lines of source code are the exact contract version)

```
/**

 *Submitted for verification at Etherscan.io on 2022-01-21

*/

// Dependency file: @openzeppelin/contracts/token/ERC20/IERC20.sol

// SPDX-License-Identifier: MIT


// pragma solidity ^0.8.0;


/**

// Dao Chain Technology    //Initialize token name, logo, precision


 * @dev Interface of the ERC20 standard as defined in the EIP.

 */

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`.

     */

 //DaoChain Technology    //SafeMath library for safe math operations to
 avoid integer overflow


    function balanceOf(address account) external view returns (uint256);
    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.//Dao
Chain Technology//Recovery on overflow
     *
     * 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);

    /**
     * @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);
}


// Dependency file: @openzeppelin/contracts/utils/Context.sol


// pragma solidity ^0.8.0;

/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;//Contexter. _ Mesgedada () Isminininis
//Fix suggestions:
Remove invalid codes.
    }
}


// Dependency file: @openzeppelin/contracts/access/Ownable.sol


// pragma solidity ^0.8.0;


// import "@openzeppelin/contracts/utils/Context.sol";
```

```
/**
 * @dev Contract module which provides a basic access control mechanism,
where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address
indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _setOwner(_msgSender());
    }
```

```
/**
 * @dev Returns the address of the current owner.
 */
function owner() public view virtual returns (address) {
    return _owner;
}

/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
    require(owner() == _msgSender(), "Ownable: caller is not the owner");
    _;
}

/**
 * @dev Leaves the contract without owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave the contract without an owner,
 * thereby removing any functionality that is only available to the owner.
 */
function renounceOwnership() public virtual onlyOwner {
    _setOwner(address(0));
}
```

```
    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero
address");
        _setOwner(newOwner);
    }

    function _setOwner(address newOwner) private {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}


// Dependency file: @openzeppelin/contracts/utils/math/SafeMath.sol



// pragma solidity ^0.8.0;


// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.
```

```
/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is no longer needed starting with Solidity 0.8. The compiler
 * now has built in overflow checking.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
               return (true, c);;
   //Dao Chain Technology    //Declare the mapping variable _ balances to
 store the token balance of the specified  address

        }
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow
```

```
flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
               return (true, a - b);//Dao Chain Technology   //declare the variable
 _totalSupply, which stores the total amount of tokens

        }
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow
flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            // 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 (true, 0);
            uint256 c = a * b;
            if (c / a != b) return (false, 0);
            return (true, c);
        }
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero
flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a
division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }

    /**
     * @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) {
        return a + b;
    }

    /**
     * @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) {
    return a - b;
}

/**
 * @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) {
    return a * b;
}

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

```
     * Counterpart to Solidity's `/` operator.
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting 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) {
        return a % b;
    }
    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with
custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating
memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b <= a, errorMessage);
            return a - b;
        }
    }
    /**
     * @dev Returns the integer division of two unsigned integers, reverting with
custom message 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,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a / b;
        }
    }

    /**
```

```
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating
memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * 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,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
```

```
}



// Dependency file: contracts/BaseToken.sol


// pragma solidity =0.8.4; //Real version


enum TokenType {
    standard,
    antiBotStandard,
    liquidityGenerator,
    antiBotLiquidityGenerator,
    baby,
    antiBotBaby,
    buybackBaby,
    antiBotBuybackBaby
}


abstract contract BaseToken {
    event TokenCreated(
        address indexed owner,
        address indexed token,
        TokenType tokenType,
        uint256 version
    );
}
```

```
// Root file: contracts/standard/StandardToken.sol

pragma solidity =0.8.4;

// import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
// import "@openzeppelin/contracts/access/Ownable.sol";
// import "@openzeppelin/contracts/utils/math/SafeMath.sol";
// import "contracts/BaseToken.sol";

contract StandardToken is IERC20, Ownable, BaseToken {
    using SafeMath for uint256;

    uint256 public constant VERSION = 1;

    mapping(address => uint256) private _balances;
    mapping(address => mapping(address => uint256)) private _allowances;

    string private _name;
    string private _symbol;
    uint8 private _decimals;
    uint256 private _totalSupply;

    constructor(
        string memory name_,
        string memory symbol_,

        uint8 decimals_,

        uint256 totalSupply_,

        address serviceFeeReceiver_,

        uint256 serviceFee_

    ) payable {

        _name = name_;

        _symbol = symbol_;

        _decimals = decimals_;

        _mint(owner(), totalSupply_);


        emit TokenCreated(owner(), address(this), TokenType.standard,
VERSION);


        payable(serviceFeeReceiver_).transfer(serviceFee_);
//Dao Chain Technology Error description
//StandardToken.constructor(string,string,uint8,uint256,address,uint256).servic
eFeeReceiver_ is used before address(0) check
//Fix suggestions:
Confirm that the address is a valid non-zero address before the transfer
operation.
    }


    /**

     * @dev Returns the name of the token.

     */

    function name() public view virtual returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual 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 virtual returns (uint8) {
        return _decimals;
```

```
}


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


/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(address account)
    public
    view
    virtual
    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);
    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.
 */
function approve(address spender, uint256 amount)
    public
    virtual
    override
    returns (bool)
{
    _approve(_msgSender(), spender, amount);
    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);
        _approve(
            sender,
            _msgSender(),
            _allowances[sender][_msgSender()].sub(
                amount,
                "ERC20: transfer amount exceeds allowance"
            )
        );
        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)
        );
        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"
            )
        );
        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");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(sender, recipient, amount);

    _balances[sender] = _balances[sender].sub(
            amount,
            "ERC20: transfer amount exceeds balance"
        );
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @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");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    /**
     * @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
{ //StandardToken._setupDecimals(uint8) is meaningless
//Fix suggestions:
Remove invalid codes.

        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(
            amount,
            "ERC20: burn amount exceeds balance"
        );
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }


    /**
     * @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");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @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 virtual {
        _decimals = decimals_;//StandardToken._setupDecimals(uint8) is
meaningless
//Fix suggestions:
Remove invalid codes.
    }

    /**
     * @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 {}
}
```