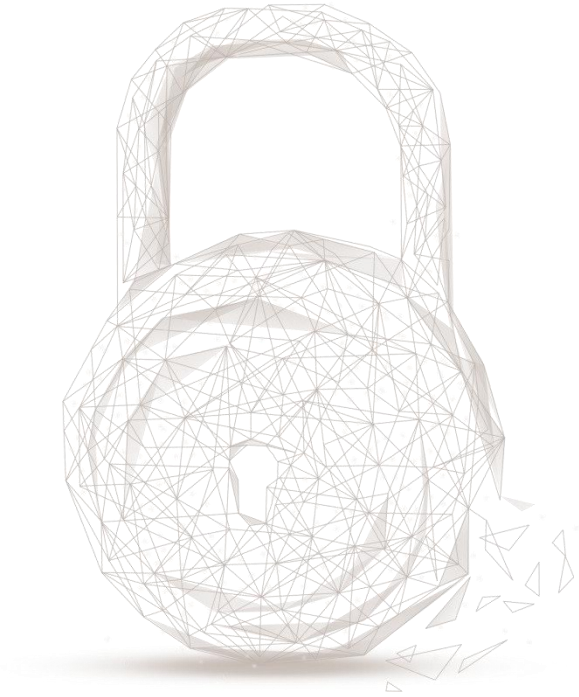
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202101071158

Smart Contract Name：

SDCToken (SDC)

Smart Contract Address：

0xe2f45b8fbcb2b5bb544fe9f796bcfeaa3a4dcdbf

Smart Contract Address Link：

https://scan.hecochain.com/address/0xe2f45b8fbcb2b5bb544fe9f796bcfeaa3a4dcdbf#contracts

Start Date：2021.01.05

Completion Date：2021.01.07

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	-	N/A
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 SDC, including Coding Standards, Security, and Business Logic. SDC **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	Stable Coin
Token symbol	SDC
decimals	18
totalSupply	Mintable without cap
Token type	ERC20

Table 1 – Basic Token Information

2、Token Vesting Information

N/A

3、Other function description

- rebase Function

As shown in the figure below, the contract implements the *rebase* function to dynamically adjust the holder's account balance. The owner address of the contract can set a *_RebaseInvoker* address, and only this address can call the *rebase* function to modify the *_perShareAmount* parameter. The user's balance is positively correlated with *_perShareAmount*.

```
314     function rebase(uint256 epoch, uint256 numerator, uint256 denominator) external onlyRebaseInvoker returns (uint256) {
315         uint256 newPerShareAmount = _perShareAmount.mul(numerator).div(denominator);// Beosin (Chengdu LianAn) // According to the previ
316         if (newPerShareAmount == 0) { // Beosin (Chengdu LianAn) // The 0 value check for 'newPerShareAmount'.
317             newPerShareAmount = 1;
318         }
319         emit Rebase(epoch, _perShareAmount, newPerShareAmount); // Beosin (Chengdu LianAn) // Trigger the event 'Rebase'.
320         _perShareAmount = newPerShareAmount; // Beosin (Chengdu LianAn) // Alter '_perShareAmount' to 'newPerShareAmount'.
321         return _perShareAmount;
322     }
```

Figure 1 *rebase* function source code

- balanceOf Function

As shown in the figure below, the contract implements the IERC20 standard interface *balanceOf* to indicate user balance. The user's final balance is equal to the user's '*_shares*' balance multiplied by '*_perShareAmount*', where the '*_shares*' balance is only changed during transfer and minting, and the change of '*_perShareAmount*' depends only on the rebase function.

```
254     function balanceOf(address account) external view returns (uint256) {
255         return _shares[account].mul(_perShareAmount);
256     }
```

Figure 2 balanceOf function source code

Audited Source Code with Comments:

```
// SPDX-License-Identifier: MIT

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
// 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;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol
```

```
pragma solidity ^0.5.0;

/**
 * @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.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be aplied to your functions to restrict their use to
 * the owner.
 */
contract Ownable {
    address private _owner; // Beosin (Chengdu LianAn) // Declare the variable '_owner' to store the address of the contract owner.

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); // Beosin (Chengdu LianAn) // Declare the event 'OwnershipTransferred'.

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    // Beosin (Chengdu LianAn) // Constructor, set owner as the address of deploying this contract.
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
```

```
     */
    // Beosin (Chengdu LianAn) // Modifier, requires that the function caller must be owner.
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0)); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        _owner = address(0); // Set owner as zero to renounce ownership.
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner); // Beosin (Chengdu LianAn) // Call the internal function '_transferOwnership', complete the 'transferOwnership' operation.
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
```

```
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'newOwner'.
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner; // Beosin (Chengdu LianAn) // Transfer owner permissions to 'newOwner'.
    }
}

// File: Token.sol

pragma solidity >=0.5.0;



contract Token is Ownable {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    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.

    address private _mintInvoker; // Beosin (Chengdu LianAn) // Declare the variable '_mintInvoker' for storing the mint invoker.
    address private _rebaseInvoker; // Beosin (Chengdu LianAn) // Declare the variable '_rebaseInvoker' for storing the rebase invoker.
    uint256 private _perShareAmount; // Beosin (Chengdu LianAn) // Declare the variable '_perShareAmount' for storing the per share amount.
    uint256 private _totalShares; // Beosin (Chengdu LianAn) // Declare the variable '_totalShares' for storing the total shares.

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

Declare the mapping variable '_allowedShares' for storing the allowance shares between two addresses.

```
    event Transfer(address indexed from, address indexed to, uint256 amount); // Beosin (Chengdu LianAn) // Declare the event 'Transfer'.
    event Approval(address indexed owner, address indexed spender, uint256 amount); // Beosin (Chengdu LianAn) // Declare the event 'Approval'.
    event Rebase(uint256 indexed epoch, uint256 oldPerShareAmount, uint256 newPerShareAmount); // Beosin (Chengdu LianAn) // Declare the event 'Rebase'.
    event RebaseInvokerChanged(address indexed previousOwner, address indexed newOwner); // Beosin (Chengdu LianAn) // Declare the event 'RebaseInvokerChanged'.
    event MintInvokerChanged(address indexed previousOwner, address indexed newOwner); // Beosin (Chengdu LianAn) // Declare the event 'MintInvokerChanged'.

    // Beosin (Chengdu LianAn) // Modifier, requires that the function caller must be '_rebaseInvoker'.
    modifier onlyRebaseInvoker() {
        require(msg.sender == _rebaseInvoker, "Rebase: caller is not the rebase invoker");
        _;
    }

    // Beosin (Chengdu LianAn) // Modifier, requires that the function caller must be '_mintInvoker'.
    modifier onlyMintInvoker() {
        require(msg.sender == _mintInvoker, "Mint: caller is not the mint invoker");
        _;
    }

    constructor(string memory name, string memory symbol, uint8 decimals, uint256 perShareAmount) public {
        _name = name; // Beosin (Chengdu LianAn) // Set the token name.
        _symbol = symbol; // Beosin (Chengdu LianAn) // Set the token symbol.
        _decimals = decimals; // Beosin (Chengdu LianAn) // Set the token decimals.
        _perShareAmount = perShareAmount; // Beosin (Chengdu LianAn) // Set the initial '_perShareAmount'.
    }

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

    function symbol() public view returns (string memory) {
```

```
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function perShareAmount() public view returns (uint256) {
        return _perShareAmount;
    }

    function totalSupply() external view returns (uint256) {
        return _totalShares.mul(_perShareAmount);
    }

    function totalShares() public view returns (uint256) {
        return _totalShares;
    }

    function mintInvoker() public view returns (address) {
        return _mintInvoker;
    }

    function rebaseInvoker() public view returns (address) {
        return _rebaseInvoker;
    }

    function balanceOf(address account) external view returns (uint256) {
        return _shares[account].mul(_perShareAmount);
    }

    function transfer(address recipient, uint256 amount) external returns (bool) {
        uint256 share = amount.div(_perShareAmount); // Beosin (Chengdu LianAn) // Calculate the number of'_shares' that need to be transferred according to the transfer amount.
        _shares[msg.sender] = _shares[msg.sender].sub(share); // Beosin (Chengdu LianAn) // Decrease the sender's '_shares' balance.
        _shares[recipient] = _shares[recipient].add(share); // Beosin (Chengdu LianAn) // Increase the recipient's '_shares' balance.
```

```
        emit Transfer(msg.sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
        return true;
    }

    function allowance(address owner, address spender) external view returns (uint256) {
        if (_allowedShares[owner][spender] >= uint256(-1).div(_perShareAmount)) { // Beosin (Chengdu LianAn) // Overflow prevention.
            return uint256(-1);
        }
        return _allowedShares[owner][spender].mul(_perShareAmount);
    }

    // Beosin (Chengdu LianAn) // Beware that changing an '_allowanceShare' with this method brings the risk that Someone may use both the old and the new '_allowanceShare' by unfortunate transaction ordering. It is recommended to set 0 before authorization.
    function approve(address spender, uint256 amount) external returns (bool) {
        _allowedShares[msg.sender][spender] = amount.div(_perShareAmount); // Beosin (Chengdu LianAn) // Alter the '_allowedShares' variable to update the caller's authorization for 'spender'.
        emit Approval(msg.sender, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool) {
        uint256 share = amount.div(_perShareAmount); // Beosin (Chengdu LianAn) // Calculate the number of'_shares' that need to be transferred according to the transfer amount.
        _allowedShares[sender][msg.sender] = _allowedShares[sender][msg.sender].sub(share); // Beosin (Chengdu LianAn) // Alter the '_allowedShares' variable to update the caller's authorization for 'spender'.
        _shares[sender] = _shares[sender].sub(share); // Beosin (Chengdu LianAn) // Decrease the sender's '_shares' balance.
        _shares[recipient] = _shares[recipient].add(share); // Beosin (Chengdu LianAn) // Increase the recipient's '_shares' balance.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
        return true;
    }

    // Beosin (Chengdu LianAn) // Alter '_rebaseInvoker' address.
```

```
    function changeRebaseInvoker(address newInvoker) public onlyOwner {
        require(newInvoker != address(0), "Rebase: new invoker is the zero address");
        emit RebaseInvokerChanged(_rebaseInvoker, newInvoker);
        _rebaseInvoker = newInvoker;
    }

    function rebase(uint256 epoch, uint256 numerator, uint256 denominator) external onlyRebaseInvoker returns (uint256) {
        uint256 newPerShareAmount = _perShareAmount.mul(numerator).div(denominator);// Beosin (Chengdu LianAn) // According to the previous value of '_perShareAmount', 'newPerShareAmount' is converted.
        if (newPerShareAmount == 0) { // Beosin (Chengdu LianAn) // The 0 value check for 'newPerShareAmount'.
            newPerShareAmount = 1;
        }
        emit Rebase(epoch, _perShareAmount, newPerShareAmount); // Beosin (Chengdu LianAn) // Trigger the event 'Rebase'.
        _perShareAmount = newPerShareAmount; // Beosin (Chengdu LianAn) // Alter '_perShareAmount' to 'newPerShareAmount'.
        return _perShareAmount;
    }

    // Beosin (Chengdu LianAn) // Alter '_mintInvoker' address.
    function changeMintInvoker(address newInvoker) public onlyOwner {
        require(newInvoker != address(0), "Mint: new invoker is the zero address");
        emit MintInvokerChanged(_mintInvoker, newInvoker);
        _mintInvoker = newInvoker;
    }

    function mint(address to, uint256 share) external onlyMintInvoker {
        require(to != address(0), "mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'to'.
        _totalShares = _totalShares.add(share); // Beosin (Chengdu LianAn) // Update '_totalShares'.
        _shares[to] = _shares[to].add(share); // Beosin (Chengdu LianAn) // Increase the '_shares' balance for 'to'.
        emit Transfer(address(0), to, share.mul(_perShareAmount)); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }
```

```
}

// Beosin (Chengdu LianAn) // Recommend the main contract to inherit 'Pausable' module to grant owner the authority of pausing all transactions when serious issue occurred.
```

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com