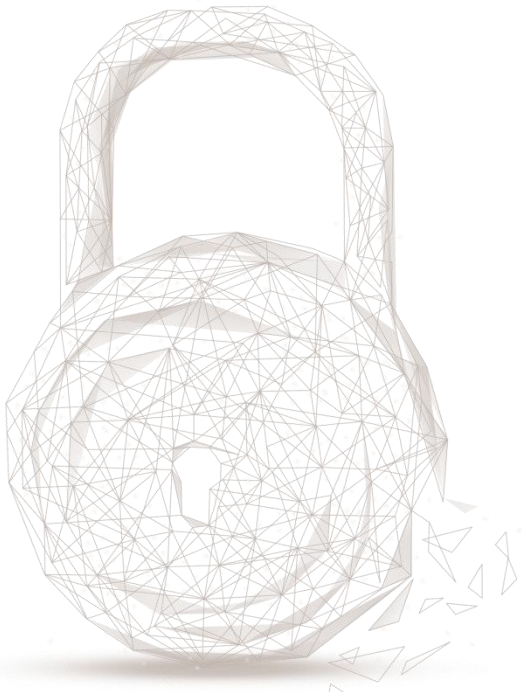
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202101071156

审计合约名称：

SDCToken（SDC）

审计合约地址：

0xe2f45b8fbcb2b5bb544fe9f796bcfeaa3a4dcdbf

审计合约链接地址：

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

合约审计开始日期：2021.01.05

合约审计完成日期：2021.01.07

审计结果：通过（优）

审计团队：成都链安科技有限公司

审计类型及结果：

序号	审计类型	审计子项	审计结果
1	代码规范审计	ERC20 Token 标准规范审计	通过
		编译器版本安全审计	通过
		可见性规范审计	通过
		gas 消耗审计	通过
		SafeMath 功能审计	通过
		fallback 函数使用审计	通过
		tx.origin 使用审计	通过
		弃用项审计	通过
		冗余代码审计	通过
		变量覆盖审计	通过
2	函数调用审计	函数调用权限审计	通过
		call/delegatecall 安全审计	通过
		返回值安全审计	通过
		自毁函数安全审计	通过
3	业务安全审计	owner 权限审计	通过
		业务逻辑审计	通过
		业务实现审计	通过
4	整型溢出审计	-	通过
5	可重入攻击审计	-	通过
6	异常可达状态审计	-	通过
7	交易顺序依赖审计	-	通过
8	块参数依赖审计	-	通过
9	伪随机数生成审计	-	通过

10	拒绝服务攻击审计	-	通过
11	代币锁仓审计	-	无锁仓
12	假充值审计	-	通过
13	event 安全审计	-	通过

备注：审计意见及建议请见代码注释。

免责声明：本次审计仅针对本报告载明的审计类型及结果表中给定的审计类型范围进行审计，其他未知安全漏洞不在本次审计责任范围之内。成都链安科技仅根据本报告出具前已经存在或发生的攻击或漏洞出具本报告，对于出具以后存在或发生的新的攻击或漏洞，成都链安科技无法判断其对智能合约安全状况可能的影响，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于合约提供者在本报告出具前已向成都链安科技提供的文件和资料，且该部分文件和资料不存在任何缺失、被篡改、删减或隐瞒的前提下作出的；如提供的文件和资料存在信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符等情况或提供文件和资料在本报告出具后发生任何变动的，成都链安科技对由此而导致的损失和不利影响不承担任何责任。成都链安科技出具的本审计报告系根据合约提供者提供的文件和资料依靠成都链安科技现掌握的技术而作出的，由于任何机构均存在技术的局限性，成都链安科技作出的本审计报告仍存在无法完整检测出全部风险的可能性，成都链安科技对由此产生的损失不承担任何责任。

本声明最终解释权归成都链安科技所有。

审计结果说明：

本公司采用形式化验证、静态分析、动态分析、典型案例测试和人工审核的方式对智能合约SDC的代码规范性、安全性以及业务逻辑三个方面进行多维度全面的安全审计。**经审计，SDC合约通过所有检测项，合约审计结果为通过(优)，合约可正常使用。**以下为本合约基本信息。

1、代币基本信息

Token name	Stable Coin
Token symbol	SDC
decimals	18
totalSupply	可铸币，无上限
Token type	ERC20

表1 代币基本信息

2、代币锁仓信息

无锁仓

3、其它函数功能描述

- rebase函数

如下图所示，合约实现了rebase函数用以动态调整持有者的账户余额。合约的owner地址可以设置一个RebaseInvoker地址，仅此地址可以调用rebase函数，进行_perShareAmount参数的修改。用户的余额与_perShareAmount正相关。

```
322     function rebase(
323         uint256 epoch,
324         uint256 numerator,
325         uint256 denominator
326     ) external onlyRebaseInvoker returns (uint256) {
327         uint256 newPerShareAmount = _perShareAmount.mul(numerator).div(
328             denominator
329         );
330         emit Rebase(epoch, _perShareAmount, newPerShareAmount);
331         _perShareAmount = newPerShareAmount;
332         return _perShareAmount;
333     }
```

图 1 rebase函数源码截图

- balanceOf函数

如下图所示，合约实现了IERC20的标准接口balanceOf用以表示用户余额。用户的最终余额等于用户的‘_shares’余额乘以‘_perShareAmount’，其中‘_shares’余额仅在转账、铸币时发生改变，‘_perShareAmount’的改变，仅取决于rebase函数。

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

图 2 balanceOf函数源码截图

合约源代码审计注释：

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

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

pragma solidity ^0.5.0; // 成都链安 // 建议固定编译器版本

/**
 * @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.
 */
// 成都链安 // SafeMath 库，用于安全数学运算以避免整型溢出
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; // 成都链安 // 声明_owner 变量,用于存储合约所有者地址

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); // 成都链安 // 声明 owner 权限转移事件

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    // 成都链安 // 构造函数，设置合约初始 owner 为合约部署者地址
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner); // 成都链安 // 触发 OwnershipTransferred 事件
    }

    /**
     * @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.
     */
    // 成都链安 // 修饰器，要求被修饰函数调用者必须为 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)); // 成都链安 // 触发 OwnershipTransferred 事件
        _owner = address(0); // 成都链安 // 设置 owner 为零地址，放弃 owner 权限
    }

    /**
     * @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); // 成都链安 // 调用内部函数_transferOwnership，完成转移 owner 权限操作
    }

    /**
     * @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"); // 成都链安 // 检查 newOwner 地址不为 0
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner; // 成都链安 // 转移 owner 权限至 newOwner
    }
}

// File: Token.sol

pragma solidity >=0.5.0;
```

```
contract Token is Ownable {
    using SafeMath for uint256; // 成都链安 // 引入 SafeMath 安全数学运算库，避免数学运算整型溢出

    string private _name; // 成都链安 // 声明变量_name，存储代币名称
    string private _symbol; // 成都链安 // 声明变量_symbol，存储代币简称
    uint8 private _decimals; // 成都链安 // 声明变量_decimals，存储代币精度

    address private _mintInvoker; // 成都链安 // 声明变量_mintInvoker，存储可调用 mint 函数的地址
    address private _rebaseInvoker; // 成都链安 // 声明变量_rebaseInvoker，存储可调用 rebase 函数的地址
    uint256 private _perShareAmount; // 成都链安 // 声明变量_perShareAmount，存储每 Share 对应的代币数量
    uint256 private _totalShares; // 成都链安 // 声明变量_totalShares，存储总的 Share 数量

    mapping(address => uint256) private _shares; // 成都链安 // 声明 mapping 变量_shares，存储指定地址的 Shares 余额
    mapping(address => mapping(address => uint256)) private _allowedShares; // 成都链安 // 声明 mapping 变量_allowedShares，存储对应地址间的授权值

    event Transfer(address indexed from, address indexed to, uint256 amount); // 成都链安 // 声明代币转账事件
    event Approval(address indexed owner, address indexed spender, uint256 amount); // 成都链安 // 声明代币授权事件
    event Rebase(uint256 indexed epoch, uint256 oldPerShareAmount, uint256 newPerShareAmount); // 成都链安 // 声明修改代币余额事件
    event RebaseInvokerChanged(address indexed previousOwner, address indexed newOwner); // 成都链安 // 声明转移 RebaseInvoker 权限事件
    event MintInvokerChanged(address indexed previousOwner, address indexed newOwner); // 成都链安 // 声明转移 MintInvokerChanged 权限事件

    // 成都链安 // 修饰器，要求被修饰函数调用者必须为_rebaseInvoker
    modifier onlyRebaseInvoker() {
        require(msg.sender == _rebaseInvoker, "Rebase: caller is not the rebase invoker");
        _;
```

```
}

// 成都链安 // 修饰器，要求被修饰函数调用者必须为_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; // 成都链安 // 设置代币名称
    _symbol = symbol; // 成都链安 // 设置代币简称
    _decimals = decimals; // 成都链安 // 设置代币精度
    _perShareAmount = perShareAmount; // 成都链安 // 设置合约初始_perShareAmount 值
}

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); // 成都链安 // 根据转账金额计算出需要转账的'_shares'数量
        _shares[msg.sender] = _shares[msg.sender].sub(share); // 成都链安 // 减少发送者的'_shares'余额
        _shares[recipient] = _shares[recipient].add(share); // 成都链安 // 增加接收者的'_shares'余额
        emit Transfer(msg.sender, recipient, amount); // 成都链安 // 触发'Transfer'事件
        return true;
    }

    function allowance(address owner, address spender) external view returns (uint256) {
        if (_allowedShares[owner][spender] >= uint256(-1).div(_perShareAmount)) { // 成都链安 // 防溢出检查
            return uint256(-1);
        }
        return _allowedShares[owner][spender].mul(_perShareAmount);
    }

    // 成都链安 // 用户调用该函数修改授权值时，可能导致多重授权，建议先将授权值重置为 0，再进行新的授权。
    function approve(address spender, uint256 amount) external returns (bool) {
        _allowedShares[msg.sender][spender] = amount.div(_perShareAmount);// 成都链安 // 修改'_allowedShares'变量，更新调用者对'spender'的授权
        emit Approval(msg.sender, spender, amount); // 成都链安 // 触发'Approval'事件
```

```
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool) {
        uint256 share = amount.div(_perShareAmount); // 成都链安 // 根据转账金额计算出需要转账的'_shares'数量
        _allowedShares[sender][msg.sender] = _allowedShares[sender][msg.sender].sub(share); // 成都链安 // 修改'_allowedShares'变量，更新调用者对'spender'的授权
        _shares[sender] = _shares[sender].sub(share); // 成都链安 // 减少发送者的'_shares'余额
        _shares[recipient] = _shares[recipient].add(share); // 成都链安 // 增加接收者的'_shares'余额
        emit Transfer(sender, recipient, amount); // 成都链安 // 触发'Transfer'事件
        return true;
    }

    // 成都链安 // 修改_rebaseInvoker 地址
    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); // 成都链安 // 根据前一个_perShareAmount 的值，换算出 newPerShareAmount。
        if (newPerShareAmount == 0) { // 成都链安 // 检查 newPerShareAmount 不为 0
            newPerShareAmount = 1;
        }
        emit Rebase(epoch, _perShareAmount, newPerShareAmount); // 成都链安 // 触发'Rebase'事件
        _perShareAmount = newPerShareAmount; // 成都链安 // 修改_perShareAmount 的值
        return _perShareAmount;
    }

    // 成都链安 // 修改_mintInvoker 地址
    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"); // 成都链安 // to 地址非 0 检查
        _totalShares = _totalShares.add(share); // 成都链安 // 更新 shares 总量
        _shares[to] = _shares[to].add(share); // 成都链安 // 修改 to 地址的 shares 余额
        emit Transfer(address(0), to, share.mul(_perShareAmount)); // 成都链安 // 触发 Transfer 事件
    }
}

// 成都链安 // 建议主合约继承 Pausable 模块，当出现重大异常时 owner 可以暂停所有交易
```

成都链安

BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号