



智能合约安全审计报告



审计编号： 202309101040

审计合约名称：

SMP (SMP)

审计合约地址：

0x1931CAFa31495741f33D89112fFB80ba60986F49

审计合约链接地址：

<https://bscscan.com/address/0x1931CAFa31495741f33D89112fFB80ba60986F49>

合约审计开始日期： 2023. 08. 25

合约审计完成日期： 2023. 09. 10

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	BEP-20 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
		能量消耗审计	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：本报告系针对项目代码而作出，本报告的任何描述、表达或措辞均不得被解释为对项目的认可、肯定或确认。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SMP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，SMP合约通过所有检测，合约审计结果为通过(优)。**以下为本合约基本信息。

1、代币基本信息

Token name	SMP
Token symbol	SMP
decimals	18
totalSupply	210,000,000 (总量恒定)
Token type	BEP-20

表1 SMP代币基本信息

2、代币锁仓信息

无锁仓

合约源代码审计注释:

Token.sol

```
// 0.5.1-c8a2
// Enable optimization
pragma solidity ^0.8.4; // 成都链安 // 建议固定编译器版本

import "./BEP20.sol";
import "./BEP20Detailed.sol";

/**
 * @title SimpleToken
 * @dev Very simple BEP20 Token example, where all tokens are pre-assigned to the
creator.
 * Note they can later distribute these tokens as they wish using `transfer` and other
 * `BEP20` functions.
 */
contract Token is BEP20, BEP20Detailed {
    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    // 成都链安 // 初始化代币名称, 标识, 精度

    constructor () public BEP20Detailed("SMP", "SMP", 18) {
        _mint(msg.sender, 210000000 * (10 ** uint256(decimals()))); // 成都链安 // 将
初始化 代币发送给合约创建者
    }
}
// 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
```

BEP20.sol

```
pragma solidity ^0.8.4;

import "./IBEP20.sol";
import "./SafeMath.sol";

/**
 * @dev Implementation of the {IBEP20} 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 {BEP20Mintable}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-BEP20-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 BEP20 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 {IBEP20-approve}.
*/
contract BEP20 is IBEP20 {
    using SafeMath for uint256; // 成都链安 // 引用 SafeMath 安全库，用于安全数学运算

    mapping (address => uint256) private _balances; // 成都链安 // 声明 mapping 变量
    _balances, 存储指定地址的代币余额

    mapping (address => mapping (address => uint256)) private _allowances; // 成都链安
    // 声明 mapping 变量_allowances, 存储对应地址间的授权值

    uint256 private _totalSupply; // 成都链安 // 声明变量_totalSupply, 存储代币总量

    /**
     * @dev See {IBEP20-totalSupply}.
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply; // 成都链安 // 返回代币总量
    }

    /**
     * @dev See {IBEP20-balanceOf}.
     */
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account]; // 成都链安 // 返回账户代币余额
    }

    /**
     * @dev See {IBEP20-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) {
        _transfer(msg.sender, recipient, amount); // 成都链安 // 调用内部函数_transfer
```


进行代币转账

```
        return true;
    }

    /**
     * @dev See {IBEP20-allowance}.
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender]; // 成都链安 // 返回指定地址 owner 对 spender
的授权值
    }

    /**
     * @dev See {IBEP20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    // 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，建议用户使用
increaseAllowance 与 decreaseAllowance 修改授权值
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    /**
     * @dev See {IBEP20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {BEP20};
     *
     * 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); // 成都链安 // 调用内部函数_transfer 进行
代币转账
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount)); // 成
都链安 // 调用内部函数_approve 更新代币发送者 sender 对调用者的授权值
        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 {IBEP20-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));
// 成都链安 // 调用内部函数_approve 增加调用者对 spender 的授权值
        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 {IBEP20-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)); // 成都链安 // 调用内部函数
        _approve 减少调用者对 spender 的授权值
        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), "BEP20: transfer from the zero address"); // 成都链安 // sender 非零地址检查
    require(recipient != address(0), "BEP20: transfer to the zero address"); // 成都链安 // recipient 非零地址检查

    _balances[sender] = _balances[sender].sub(amount); // 成都链安 // 更新 sender 地址代币余额
    _balances[recipient] = _balances[recipient].add(amount); // 成都链安 // 更新 recipient 地址代币余额
    emit Transfer(sender, recipient, amount); // 成都链安 // 触发 Transfer 事件
}

/** @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), "BEP20: mint to the zero address"); // 成都链安 // account 非零地址检查

    _totalSupply = _totalSupply.add(amount); // 成都链安 // 更新代币总量
    _balances[account] = _balances[account].add(amount); // 成都链安 // 更新 account 地址代币余额
    emit Transfer(address(0), account, amount); // 成都链安 // 触发 Transfer 事件
}

/**
 * @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 value) internal {
    require(account != address(0), "BEP20: burn from the zero address"); // 成都链安
// account 非零地址检查

    _totalSupply = _totalSupply.sub(value); // 成都链安 // 更新代币总量
    _balances[account] = _balances[account].sub(value); // 成都链安 // 更新 account
地址代币余额
    emit Transfer(account, address(0), value); // 成都链安 // 触发 Transfer 事件
}

/**
 * @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), "BEP20: approve from the zero address"); // 成都链
安 // owner 非零地址检查
    require(spender != address(0), "BEP20: approve to the zero address"); // 成都链
安 // spender 非零地址检查

    _allowances[owner][spender] = value; // 成都链安 // 更新 owner 对 spender 的授权值
    emit Approval(owner, spender, value); // 成都链安 // 触发 Approval 事件
}

/**
 * @dev Destroys `amount` tokens from `account`. `amount` is then deducted
 * from the caller's allowance.
 *
 * See {_burn} and {_approve}.
 */
function _burnFrom(address account, uint256 amount) internal {
    _burn(account, amount); // 成都链安 // 销毁账户代币
    _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount)); //
成都链安 // 更新授权值
}
}
```

BEP20Detailed.sol

```
pragma solidity ^0.8.4;

import "./IBEP20.sol";

/**
 * @dev Optional functions from the BEP20 standard.
 */
contract BEP20Detailed is IBEP20 {
    string private _name; // 成都链安 // 声明变量_name, 用于存储代币名称
    string private _symbol; // 成都链安 // 声明变量_symbol, 用于存储代币标识
    uint8 private _decimals; // 成都链安 // 声明变量_decimals, 用于存储代币精度

    /**
     * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
     * these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name; // 成都链安 // 初始化代币名称
        _symbol = symbol; // 成都链安 // 初始化代币标识
        _decimals = decimals; // 成都链安 // 初始化代币精度
    }

    /**
     * @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

```

```
    * {IBEP20-balanceOf} and {IBEP20-transfer}.
```

```
    */
```

```
function decimals() public view returns (uint8) {
```

```
    return _decimals;
```

```
}
```

```
}
```

IBEP20.sol

```
pragma solidity ^0.8.4;
```

```
/**
```

```
 * @dev Interface of the BEP20 standard as defined in the EIP. Does not include
```

```
 * the optional functions; to access them see {BEP20Detailed}.
```

```
 */
```

```
// 成都链安 // 定义 BEP20 标准要求的接口函数与事件
```

```
interface IBEP20 {
```

```
    /**
```

```
     * @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);

/**
 * @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);
}
```

SafeMath.sol

```
pragma solidity ^0.8.4;

/**
 * @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;  
}  
}
```



成都链安
BEOSIN

官方网址

<https://lianantech.com>

电子邮箱

vaas@lianantech.com

微信公众号

