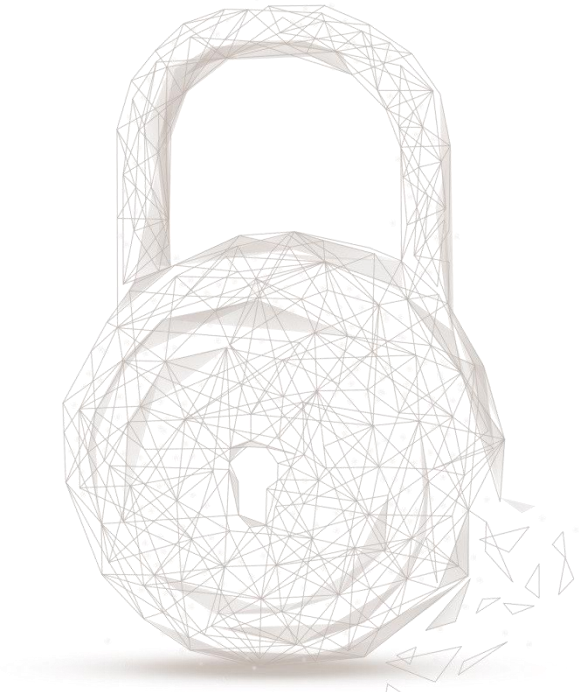
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202104081554

审计合约名称：

AppleToken（APT）

审计合约地址：

0x28F1Ec92Dd1BF76e2Bc2eE9290fb841A4547a325

合约审计开始日期：2021.04.15

合约审计完成日期：2021.04.22

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	HRC-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
		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	-	通过
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Apple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，Apple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	Apt
Token symbol	Apt
decimals	18
totalSupply	初始2亿（可铸币，最大上限20亿）
Token type	HRC-20

表1 代币基本信息

2、代币锁仓信息

如下图所示，合约所有者可调用 lock 函数进行锁仓。锁仓时，需指定锁仓地址、锁仓金额及锁仓种类，不同种类锁仓对应的日释放量不同，且可被合约所有者调控。锁仓释放最小时间单位为“一天”，不满一天不计算，因此全部锁仓释放时间可能会比预期时间长，但不影响总释放量。

```
389     function lock(address _account, uint256 _amount, uint256 _type) public onlyOwner {
390         require(_account != address(0), "Cannot transfer to the zero address");
391         require(lockedUser[_account].lockedAmount == 0, "exist locked token");
392         require(_account != swapGovContract, "equal to swapGovContract");
393         lockedUser[_account].initLock = _amount;
394         lockedUser[_account].lockedAmount = _amount;
395         lockedUser[_account].lastUnlockTs = block.timestamp >= lockreleasetime ? block.timestamp : lockreleasetime;
396         lockedUser[_account].releaseType = _type;
397         _balances[_msgSender()] = _balances[_msgSender()].sub(_amount);
398         _balances[_account] = _balances[_account].add(_amount);
399         emit Lock(_account, block.timestamp, _amount, _type);
400         emit Transfer(_msgSender(), _account, _amount);
401     }
```

图 1 lock 函数源码截图

如下图所示，代币锁仓余额需要用户手动调用 unlock 函数进行解锁，不会自动解锁。

```
403     function unlock() public {
404         uint256 amount = getAvailablelockAmount(_msgSender(), lockedUser[_msgSender()].releaseType);
405         require(amount > 0, "amount equal 0");
406         lockedUser[_msgSender()].lockedAmount = lockedUser[_msgSender()].lockedAmount.sub(amount);
407         lockedUser[_msgSender()].lastUnlockTs = block.timestamp;
408         emit UnLock(_msgSender(), block.timestamp, amount);
409     }
```

图 2 unlock 函数源码截图

- 修复建议：调用 unlock 解锁时，更改 lastUnlockTs 变量为当前已解锁的天数的时间戳，即可避免误差。
- 修复结果：忽略

3、其它函数功能描述

➢ 铸币功能

合约实现了铸币功能，合约所有者可以添加Minter权限，拥有Minter权限的地址可以调用mint函数进行铸币，铸币上限为20亿。

➢ 黑名单功能

合约实现了黑名单功能，合约所有者将有权添加地址黑名单。被添加为黑名单的地址将无法参与转账。

合约源代码审计注释：

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

pragma solidity 0.6.12;

/**
```

```
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error.
 */
// 成都链安 // SafeMath 库，用于安全数学运算以避免整型溢出
library SafeMath {
    /**
     * @dev Multiplie two unsigned integers, revert on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, revert on division by zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtract two unsigned integers, revert on underflow (i.e. if subtrahend is greater than
minuend).
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
```

```
    * @dev Add two unsigned integers, revert on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }
}


/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}
```

```
/*
 * @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 GSN 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.
 */
contract Context {

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
}

/**
 * @title ERC20 interface
 * @dev See https://eips.ethereum.org/EIPS/eip-20
 */
// 成都链安 // 定义 ERC20 标准接口
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}


/**
 * @title Standard ERC20 token
 * @dev Implementation of the basic standard token.
```

```
 */
contract StandardToken is IERC20, Context {
    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

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

    uint256 internal _totalSupply; // 成都链安 // 声明 uint256 变量_totalSupply，存储代币的总量

    /**
     * @dev Total number of tokens in existence.
     */
    function totalSupply() public override view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev Get the balance of the specified address.
     * @param owner The address to query the balance of.
     * @return A uint256 representing the amount owned by the passed address.
     */
    function balanceOf(address owner) public override view  returns (uint256) {
        return _balances[owner];
    }

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param owner The address which owns the funds.
     * @param spender The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    function allowance(address owner, address spender) public override view returns (uint256) {
        return _allowed[owner][spender];
    }

    /**
     * @dev Transfer tokens to a specified address.
     * @param to The address to transfer to.
     * @param value The amount to be transferred.
     */
    function transfer(address to, uint256 value) public virtual override returns (bool) {
        _transfer(_msgSender(), to, value); // 成都链安 // 调用内部函数_transfer 进行转账
        return true;
```

```
    }

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
     * 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
     * @param spender The address which will spend the funds.
     * @param value The amount of tokens to be spent.
     */
    // 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，建议使用 increaseAllowance 和 decreaseAllowance 修改授权值
    function approve(address spender, uint256 value) public override returns (bool) {
        _approve(_msgSender(), spender, value); // 成都链安 // 调用内部函数_approve 进行授权
        return true;
    }

    /**
     * @dev Transfer tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required as per the specification,
     * and other compliant implementations may not emit the event.
     * @param from The address which you want to send tokens from.
     * @param to The address which you want to transfer to.
     * @param value The amount of tokens to be transferred.
     */
    function transferFrom(address from, address to, uint256 value) public virtual override returns (bool) {
        _transfer(from, to, value); // 成都链安 // 调用内部函数_transfer 进行转账
        _approve(from, _msgSender(), _allowed[from][_msgSender()].sub(value)); // 成都链安 // 调用内部函数_approve 更新授权
        return true;
    }

    /**
     * @dev Increase the amount of tokens that an owner allowed to a spender.
     * approve should be called when _allowed[msg.sender][spender] == 0. To increment
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param addedValue The amount of tokens to increase the allowance by.
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowed[_msgSender()][spender].add(addedValue)); // 成都链安
```

```
// 调用内部函数_approve 进行授权
        return true;
    }

    /**
     * @dev Decrease the amount of tokens that an owner allowed to a spender.
     * approve should be called when _allowed[msg.sender][spender] == 0. To decrement
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param subtractedValue The amount of tokens to decrease the allowance by.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowed[_msgSender()][spender].sub(subtractedValue)); // 成都链安 // 调用内部函数_approve 进行授权
        return true;
    }

    /**
     * @dev Transfer tokens for a specified address.
     * @param from The address to transfer from.
     * @param to The address to transfer to.
     * @param value The amount to be transferred.
     */
    function _transfer(address from, address to, uint256 value) internal {
        require(to != address(0), "Cannot transfer to the zero address"); // 成都链安 // 非零地址检查
        _balances[from] = _balances[from].sub(value); // 成都链安 // 减少 from 地址的代币余额
        _balances[to] = _balances[to].add(value); // 成都链安 // 增加 to 地址的代币余额
        emit Transfer(from, to, value); // 成都链安 // 触发 Transfer 事件
    }

    /**
     * @dev Approve an address to spend another addresses' tokens.
     * @param owner The address that owns the tokens.
     * @param spender The address that will spend the tokens.
     * @param value The number of tokens that can be spent.
     */
    function _approve(address owner, address spender, uint256 value) internal {
        require(spender != address(0), "Cannot approve to the zero address"); // 成都链安 // 非零地址检查
        require(owner != address(0), "Setter cannot be the zero address"); // 成都链安 // 非零地址检查
        _allowed[owner][spender] = value; // 成都链安 // 修改 owner 地址对 spender 地址的授权值
        emit Approval(owner, spender, value); // 成都链安 // 触发 Approval 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), "ERC20: mint to the zero address");

        _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 {
        require(account != address(0), "ERC20: burn from the zero address"); // 成都链安 // 非零地址检查
        _balances[account] = _balances[account].sub(amount); // 成都链安 // 减少 account 地址的代币余额
        _totalSupply = _totalSupply.sub(amount); // 成都链安 // 更新_totalSupply 的值
        emit Transfer(account, address(0), amount); // 成都链安 // 触发 Transfer 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 virtual {
        _burn(account, amount); // 成都链安 // 调用内部函数_burn 进行代币销毁
        _approve(account, _msgSender(), _allowed[account][_msgSender()].sub(amount)); // 成都链安 // 调用内部函数_approve 更新授权
```

```
    }

}


contract MinterRole {
    using Roles for Roles.Role;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    Roles.Role private _minters;

    constructor () internal {
        _addMinter(msg.sender);
    }

    modifier onlyMinter() {
        require(isMinter(msg.sender), "MinterRole: caller does not have the Minter role");
        _;
    }

    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }

    function addMinter(address account) public onlyMinter {
        _addMinter(account);
    }

    function renounceMinter() public {
        _removeMinter(msg.sender);
    }

    function _addMinter(address account) internal {
        _minters.add(account);
        emit MinterAdded(account);
    }

    function _removeMinter(address account) internal {
        _minters.remove(account);
        emit MinterRemoved(account);
    }
}
```

```
/**
 * @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 applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address internal _owner; // 成都链安 // 声明变量_owner，用于存储合约所有者

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); // 成都链安 // 声明 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，仅_owner 地址可以调用
    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 _msgSender() == _owner;
    }

    /**
     * @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 进行所有者权限转移
    }
```

```
    /**
     * @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 // 非零地址检查
        emit OwnershipTransferred(_owner, newOwner); // 成都链安 // 触发 OwnershipTransferred 事件
        _owner = newOwner; // 成都链安 // 修改_owner 为 newOwner
    }
}


/**
 * @dev Extension of `ERC20` that adds a set of accounts with the `MinterRole`,
 * which have permission to mint (create) new tokens as they see fit.
 *
 * At construction, the deployer of the contract is the only minter.
 */
contract ERC20Mintable is StandardToken, MinterRole {
    uint256 public constant cap = 2000000000 * (10**18); //20 亿 // 成都链安 // 声明变量 cap，用于存储铸币上限
    /**
     * @dev See `ERC20._mint`.
     *
     * Requirements:
     *
     * - the caller must have the `MinterRole`.
     */
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        require(totalSupply().add(amount)<= cap, "more than token limit"); // 成都链安 // 铸币上限检查
        _mint(account, amount); // 成都链安 // 调用内部函数进行铸币
        return true;
    }
}


contract AppleToken is ERC20Mintable, Ownable{

//----------------Token Info----------------//
    string public constant name = "Apple"; // 成都链安 // 声明常量 name，存储代币名称
    string public constant symbol = "Apple"; // 成都链安 // 声明常量 symbol，存储代币简称
    uint8 public constant decimals = 18; // 成都链安 // 声明常量 decimals，存储代币精度
    uint256 public constant INITIAL_SUPPLY = 200000000 * 10 ** 18; // 成都链安 // 声明常量 INITIAL_SUPPLY，存储代币初始铸币量
```

```
    address public swapGovContract; // 成都链安 // 声明变量 swapGovContract，用以存储治理合约地址
    // bool setGovFlag = false;

//---------------Lock Info---------------//
    uint256 public techReleaseByDay = 6040 * 10 ** 18; // 成都链安 // 声明变量 techReleaseByDay，存储“技术方”每天的解锁代币数
    uint256 public capitalReleaseByDay =64 * 10 ** 18; // 成都链安 // 声明变量 capitalReleaseByDay，存储“资本方”每天的解锁代币数
    uint256 public nodeReleaseByDay = 44 * 10 ** 18; // 成都链安 // 声明变量 nodeReleaseByDay，存储“超级节点”每天的解锁代币数
    uint256 public lockreleasetime = 1620576000; //05.10 // 成都链安 // 声明变量 lockreleasetime，存储开始释放时间
    // uint256 public lockedAmount;

    // 成都链安 // 声明结构体 LockInfo，用以存储锁仓的相关信息
    struct LockInfo {
        uint256 initLock; // 成都链安 // 声明变量 initLock，存储初始锁仓值
        uint256 lockedAmount; // 成都链安 // 声明变量 lockedAmount，存储剩余锁仓值
        uint256 lastUnlockTs; // 成都链安 // 声明变量 lastUnlockTs，存储上次解锁时间
        uint256 releaseType; // 成都链安 // 声明变量 releaseType，存储锁仓类别
    }

    mapping(address => LockInfo) public lockedUser; // 成都链安 // 声明变量 lockedUser，存储对应地址所有锁仓信息

    event Lock(address account, uint256 startTime, uint256 amount, uint256 releaseType); // 成都链安 // 声明 Lock 事件
    event UnLock(address account, uint256 unlockTime, uint256 amount); // 成都链安 // 声明 UnLock 事件
//---------------Blacklist module---------------//
    mapping(address => bool) private _isBlackListed; // 成都链安 // 声明变量_isBlackListed，用以存储黑名单地址
    event AddedBlackLists(address[]); // 成都链安 // 声明 AddedBlackLists 事件
    event RemovedBlackLists(address[]); // 成都链安 // 声明 RemovedBlackLists 事件

    constructor() public {
        _totalSupply = INITIAL_SUPPLY; // 成都链安 // 更新_totalSupply
        _balances[msg.sender] = _totalSupply; // 成都链安 // 将所有代币发送至调用者地址
        emit Transfer(address(0), msg.sender, INITIAL_SUPPLY); // 成都链安 // 触发 Transfer 事件
        _owner = msg.sender; // 成都链安 // 设置合约所有者为调用者地址
        emit OwnershipTransferred(address(0), msg.sender); // 成都链安 // 触发 OwnershipTransferred 事件
```

```
    }

    function isBlackListed(address user) public view returns (bool) {
        return _isBlackListed[user];
    }

    // 成都链安 // 批量添加黑名单，仅合约所有者可以调用
    function addBlackLists(address[] calldata _evilUser) public onlyOwner {
        for (uint i = 0; i < _evilUser.length; i++) {
            _isBlackListed[_evilUser[i]] = true;
        }
        emit AddedBlackLists(_evilUser);
    }

    // 成都链安 // 批量移除黑名单，仅合约所有者可以调用
    function removeBlackLists(address[] calldata _clearedUser) public onlyOwner {
        for (uint i = 0; i < _clearedUser.length; i++) {
            delete _isBlackListed[_clearedUser[i]];
        }
        emit RemovedBlackLists(_clearedUser);
    }



    // function lockToGov() public onlyOwner {
    //     _transfer(_owner, swapGovContract, MINEREWARD); // transfer/freeze to swapGovContract
    //     lockedAmount = lockedAmount.add(MINEREWARD);
    // }
    function lock(address _account, uint256 _amount, uint256 _type) public onlyOwner {
        require(_type > 0 && _type < 4); // 成都链安 // _type 参数检查
        require(_account != address(0), "Cannot transfer to the zero address"); // 成都链安 // 非零地址检查
        require(lockedUser[_account].lockedAmount == 0, "exist locked token"); // 成都链安 // 检查指定账户是否已经存在锁仓，若存在则不可对其新增锁仓
        require(_account != swapGovContract, "equal to swapGovContract"); // 成都链安 // 检查锁仓地址不能是治理合约地址
        lockedUser[_account].initLock = _amount; // 成都链安 // 设置锁仓信息 initLock
        lockedUser[_account].lockedAmount = _amount; // 成都链安 // 设置锁仓信息 lockedAmount
        lockedUser[_account].lastUnlockTs = block.timestamp >= lockreleasetime ? block.timestamp : lockreleasetime; // 成都链安 // 设置锁仓信息 lastUnlockTs
        lockedUser[_account].releaseType = _type; // 成都链安 // 设置锁仓信息 releaseType
        _balances[_msgSender()] = _balances[_msgSender()].sub(_amount); // 成都链安 // 减少调用者的代币余额
        _balances[_account] = _balances[_account].add(_amount); // 成都链安 // 增加锁仓地址的代币余额
        emit Lock(_account, block.timestamp, _amount, _type); // 成都链安 // 触发 Lock 事件
        emit Transfer(_msgSender(), _account, _amount); // 成都链安 // 触发 Transfer 事件
    }
```

```
    function unlock() public {
        uint256 amount = getAvailablelockAmount(_msgSender(),
lockedUser[_msgSender()].releaseType); // 成都链安 // 调用 getAvailablelockAmount 计算当前可释放的代币数量
        require(amount > 0, "amount equal 0"); // 成都链安 // amount 非零检查
        lockedUser[_msgSender()].lockedAmount =
lockedUser[_msgSender()].lockedAmount.sub(amount); // 成都链安 // 更新锁仓信息 lockedAmount
        lockedUser[_msgSender()].lastUnlockTs = block.timestamp; // 成都链安 // 更新锁仓信息 lastUnlockTs
        emit UnLock(_msgSender(), block.timestamp, amount); // 成都链安 // 触发 UnLock 事件
    }

    function getAvailablelockAmount(address account, uint256 releaseType) public view returns (uint256)
{
        if(lockedUser[account].lockedAmount == 0) { // 成都链安 // lockedAmount 非零检查
            return 0;
        }

        if(block.timestamp <= lockedUser[account].lastUnlockTs) { // 成都链安 // 检查当前时间大于上次解锁时间
            return 0;
        }

        uint256 _days = block.timestamp.sub(lockedUser[account].lastUnlockTs).div(86400); // 成都链安 // 计算当前据上次解锁的时间间隔，以“天”为单位
        if(_days > 0 && releaseType == 1) {
            uint256 _releaseAmount = _days.mul(techReleaseByDay); // 成都链安 // 根据类别计算出解锁量
            return lockedUser[account].lockedAmount > _releaseAmount ? _releaseAmount :
lockedUser[account].lockedAmount; // 成都链安 // 返回最终可解锁的值
        }

        if(_days > 0 && releaseType == 2) { // 成都链安 // 根据类别计算出解锁量
            uint256 _releaseAmount = _days.mul(capitalReleaseByDay);
            return lockedUser[account].lockedAmount > _releaseAmount ? _releaseAmount :
lockedUser[account].lockedAmount; // 成都链安 // 返回最终可解锁的值
        }

        if(_days > 0 && releaseType == 3) { // 成都链安 // 根据类别计算出解锁量
            uint256 _releaseAmount = _days.mul(nodeReleaseByDay);
            return lockedUser[account].lockedAmount > _releaseAmount ? _releaseAmount :
lockedUser[account].lockedAmount; // 成都链安 // 返回最终可解锁的值
        }
        return 0;
    }
```

```
    function transfer(address _to, uint256 _value) public override returns (bool) {
        require(!isBlackListed(_msgSender())); // 成都链安 // 调用者黑名单检查
        require(!isBlackListed(_to)); // 成都链安 // _to 地址黑名单检查
        require(_balances[_msgSender()].sub(lockedUser[_msgSender()].lockedAmount) >= _value); // 成都链安 // 调用者可用余额检查
        return super.transfer(_to, _value); // 成都链安 // 调用父合约中的 transfer 函数进行转账
    }

    function transferFrom(address _from, address _to, uint256 _value) public override returns (bool) {
        require(!isBlackListed(_msgSender())); // 成都链安 // 调用者黑名单检查
        require(!isBlackListed(_from)); // 成都链安 // _from 地址黑名单检查
        require(!isBlackListed(_to)); // 成都链安 // _to 地址黑名单检查
        require(_balances[_from].sub(lockedUser[_from].lockedAmount) >= _value); // 成都链安 // _from 地址可用余额检查
        return super.transferFrom(_from, _to, _value); // 成都链安 // 调用父合约中的 transferFrom 函数进行转账
    }

    /**
     * @dev Transfer tokens to multiple addresses.
     */
    function batchTransfer(address[] memory addressList, uint256[] memory amountList) public onlyOwner returns (bool) {
        uint256 length = addressList.length; // 成都链安 // 读取参数 addressList 的长度
        require(addressList.length == amountList.length, "Inconsistent array length"); // 成都链安 // 检查 addressList 长度与 amountList 相等
        require(length > 0 && length <= 150, "Invalid number of transfer objects"); // 成都链安 // 检查 addressList 的长度非零且不超过 150
        uint256 amount; // 成都链安 // 声明临时变量 amount，用以存储转账总量
        for (uint256 i = 0; i < length; i++) {
            require(amountList[i] > 0, "The transfer amount cannot be 0"); // 成都链安 // amountList 非零检查
            require(addressList[i] != address(0), "Cannot transfer to the zero address"); // 成都链安 // addressList 非零地址检查
            require(!isBlackListed(addressList[i])); // 成都链安 // addressList 黑名单检查
            amount = amount.add(amountList[i]); // 成都链安 // 更新临时变量 amount
            _balances[addressList[i]] = _balances[addressList[i]].add(amountList[i]); // 成都链安 // 增加 addressList[i]地址的代币余额
            emit Transfer(_msgSender(), addressList[i], amountList[i]); // 成都链安 // 触发 Transfer 事件
        }
        require(_balances[_msgSender()].sub(lockedUser[_msgSender()].lockedAmount) >= amount, "Not enough tokens to transfer"); // 成都链安 // 检查调用者可用代币余额足够
        _balances[_msgSender()] = _balances[_msgSender()].sub(amount); // 成都链安 // 检查调用者可用代币余额足够
        return true;
```

```
    }

    function burn(uint256 amount) public virtual {
        _checkBeforeBurn(_msgSender(), amount); // 成都链安 // 检查调用者可用余额足够
        _burn(_msgSender(), amount); // 成都链安 // 调用内部函数_burn 对调用者的代币进行销毁
    }

    function burnFrom(address account, uint256 amount) public virtual {
        _checkBeforeBurn(account, amount); // 成都链安 // 检查调用者可用余额足够
        _burnFrom(account, amount); // 成都链安 // 调用内部函数_burnFrom 对 account 地址的代币进行销毁
    }

    function _checkBeforeBurn(address account, uint256 amount) internal {
        uint256 amt = lockedUser[account].lockedAmount;
        if (amt > 0) {
            require(balanceOf(account).sub(amt) >= amount, "token balance no enought burn");
            }

    }



    function setGovAddr(address _swapGovContract) public onlyOwner {
        // require(!setGovFlag); // only once
        swapGovContract = _swapGovContract; // 成都链安 // 修改变量 swapGovContract 为 _swapGovContract
        // setGovFlag = true;
    }
    function setReleaseByDay(uint256 _techReleaseByDay, uint256 _capitalReleaseByDay, uint256 _nodeReleaseByDay) public onlyOwner {
        require(_techReleaseByDay > 0, " must be great 0"); // 成都链安 // _techReleaseByDay 非零检查
        require(_capitalReleaseByDay > 0, " must be great 0"); // 成都链安 // _capitalReleaseByDay 非零检查
        require(_nodeReleaseByDay > 0, " must be great 0"); // 成都链安 // _nodeReleaseByDay 非零检查
        techReleaseByDay = _techReleaseByDay; // 成都链安 // 修改 techReleaseByDay 的值为 _techReleaseByDay
        capitalReleaseByDay = _capitalReleaseByDay; // 成都链安 // 修改 capitalReleaseByDay 的值为 _capitalReleaseByDay
        nodeReleaseByDay = _nodeReleaseByDay; // 成都链安 // 修改 nodeReleaseByDay 的值为 _nodeReleaseByDay
    }
}
// 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