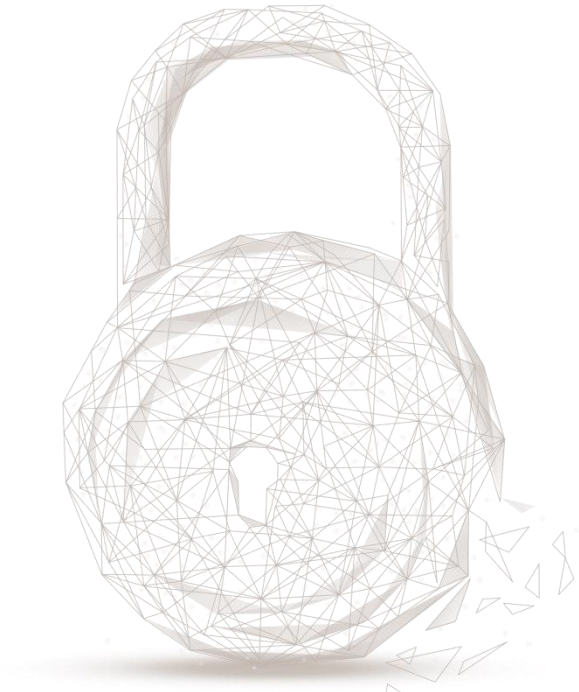
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202005281128

审计合约名称：

ClinTex (CLI)

审计合约链接地址：

https://github.com/ClinTex/CLXTokenSale/tree/develop

Commit Hash：

3677da08f4e330d9ac0b76b032d69fe15696f57d

审计合约名称：

序号	路径
1	contracts/CLIToken.sol
2	contracts/LockupContract.sol
3	contracts/managment/Management.sol
4	contracts/managment/Managed.sol
5	contracts/managment/Constants.sol

合约审计开始日期：2020.05.08

合约审计完成日期：2020.05.28

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	-	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。

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 CLI 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，CLI 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	ClinTex
Token symbol	CLI
decimals	18
totalSupply	2亿（可增发，代币上限为2亿，可销毁）
Token type	ERC20

表1 代币基本信息

2、代币锁仓信息

合约具有锁仓功能，具有锁仓权限者可以在合约部署后调用锁仓函数进行锁仓。

合约源代码审计注释：

```
// 成都链安 // 文件名称：contracts/CLIToken.sol
pragma solidity 0.5.17;
// 成都链安 // 导入合约所需的其他合约模块
import "@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./managment/Managed.sol";
import "./LockupContract.sol";
import "./CLIAllocator.sol";


contract CLIToken is ERC20, ERC20Detailed, Managed {
    // 成都链安 // 修饰器，要求调用者必须具有可允许转账的代币余额
    modifier requireUnlockedBalance(
        address _address,
        uint256 _value,
        uint256 _time,
        uint256 _holderBalance
    ) {

        require(
            LockupContract(
                management.contractRegistry(CONTRACT_LOCKUP)
            ).isTransferAllowed( // 成都链安 // 检查指定地址代币转账状态
```

```
                _address,
                _value,
                _time,
                _holderBalance
            ),
            ERROR_NOT_AVAILABLE
        );
        _;
    }
    // 成都链安 // 构造函数，初始化代币信息以及设置 Managed 合约实例
    constructor(
        address _management
    )
        public
        ERC20Detailed("ClinTex", "CLI", 18) // 成都链安 // 执行 ERC20Detailed 合约构造函数，初始化代币基本信息
        Managed(_management)
    {
    }
    // 成都链安 // 铸币函数，调用者向目标地址_account 铸造一定数量的代币
    function mint(
        address _account,
        uint256 _amount
    )
        public
        requirePermission(CAN_MINT_TOKENS) // 成都链安 // 要求调用者必须具有可铸币权限
        canCallOnlyRegisteredContract(CONTRACT_ALLOCATOR) // 成都链安 // 要求调用者必须为 CLIAllocator 合约
        returns (bool)
    {
        require(
            _amount <= CLIAllocator(
                management.contractRegistry(CONTRACT_ALLOCATOR)
            ).tokensAvailable(totalSupply()), // 成都链安 // 调用 CLIAllocator 合约的 tokensAvailable 函数获取可铸币的代币数量，要求本次铸币数量_amount 不大于可铸币代币数量
            ERROR_WRONG_AMOUNT
        );
        _mint(_account, _amount); // 成都链安 // 调用内部函数_mint 向指定地址铸币
        return true;
    }
    // 成都链安 // 代币转账函数，函数调用者向指定地址发送一定数量的代币
    function transfer(
        address _to,
        uint256 _tokens
    )
        public
        requireUnlockedBalance(
            msg.sender,
```

```
            _tokens,
            block.timestamp,
            balanceOf(msg.sender)
        ) // 成都链安 // 要求调用者必须具有可允许转账的代币余额
        returns (bool)
    {
        super.transfer(_to, _tokens); // 成都链安 // 调用父合约的 transfer 函数进行代币转账

        return true;
    }
    // 成都链安 // 代理转账函数，调用者代理源地址_holder 发送一定数量的代币至接收地址_to
    function transferFrom(
        address _holder,
        address _to,
        uint256 _tokens
    )
        public
        requireUnlockedBalance(
            _holder,
            _tokens,
            block.timestamp,
            balanceOf(_holder)
        ) // 成都链安 // 要求代理源地址_holder 必须具有可允许转账的代币余额
        returns (bool)
    {
        super.transferFrom(_holder, _to, _tokens); // 成都链安 // 调用父合约的 transferFrom
函数进行代理代币转账

        return true;
    }
    // 成都链安 // 代币销毁函数，函数调用者销毁指定数量的代币
    function burn(uint256 value)
        public
        requirePermission(CAN_BURN_TOKENS) // 成都链安 // 要求调用者必须具有代币销毁权限
        requireUnlockedBalance(
            msg.sender,
            value,
            block.timestamp,
            balanceOf(msg.sender)
        ) // 成都链安 // 要求调用者必须具有可允许销毁的代币余额
    {
        require(balanceOf(msg.sender) >= value, ERROR_WRONG_AMOUNT); // 成都链安 // 余额检
查，要求销毁数额 value 不高于函数调用者 msg.sender 的代币余额
        super._burn(msg.sender, value); // 成都链安 // 调用父合约的内部函数_burn，进行代币销
毁
    }
}
```

```
// 成都链安 // 文件名称：contracts/LockupContract.sol
pragma solidity 0.5.17;
// 成都链安 // 导入合约所需的其他合约模块
import "@openzeppelin/contracts/math/SafeMath.sol";
import "./managment/Managed.sol";


contract LockupContract is Managed {
    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

    uint256 public constant PERCENT_ABS_MAX = 100;
    bool public isPostponedStart; // 成都链安 // 声明变量 isPostponedStart，存储锁仓是否延迟开始标识
    uint256 public postponedStartDate; // 成都链安 // 声明变量 postponedStartDate，存储锁仓延迟开始时间

    mapping(address => uint256[]) public lockedAllocationData; // 成都链安 // 声明 mapping 变量 lockedAllocationData，存储指定地址的锁仓相关信息

    mapping(address => uint256) public manuallyLockedBalances; // 成都链安 // 声明 mapping 变量 manuallyLockedBalances，存储指定地址的手动锁仓代币数量

    event Lock(address holderAddress, uint256 amount); // 成都链安 // 声明锁仓事件
    // 成都链安 // 构造函数，开启锁仓延迟
    constructor(address _management) public Managed(_management) {
        isPostponedStart = true;
    }
    // 成都链安 // 检查指定地址是否允许转账
    function isTransferAllowed(
        address _address,
        uint256 _value,
        uint256 _time,
        uint256 _holderBalance
    )
    external
    view
    returns (bool)
    {
        uint256 unlockedBalance = getUnlockedBalance(
            _address,
            _time,
            _holderBalance
        ); // 成都链安 // 获取指定地址_address 未被锁定的代币余额
        if (unlockedBalance >= _value) { // 成都链安 // 如果未被锁定的代币余额不小于转账数量_value，则返回 true
            return true;
```

```
        }
        return false;
    }
    // 成都链安 // 代币锁仓分配函数
    function allocationLog(
        address _address,
        uint256 _amount,
        uint256 _startingAt,
        uint256 _lockPeriodInSeconds,
        uint256 _initialUnlockInPercent,
        uint256 _releasePeriodInSeconds
    )
        public
        requirePermission(CAN_LOCK_TOKENS) // 成都链安 // 要求调用者必须具有锁仓权限
    {
        // 成都链安 // 存储锁仓信息
        lockedAllocationData[_address].push(_startingAt);
        if (_initialUnlockInPercent > 0) { // 成都链安 // 如果初始未锁定（已释放）代币百分比大于 0
            _amount = _amount.mul(uint256(PERCENT_ABS_MAX)
                .sub(_initialUnlockInPercent)).div(PERCENT_ABS_MAX);
        }
        lockedAllocationData[_address].push(_amount);
        lockedAllocationData[_address].push(_lockPeriodInSeconds);
        lockedAllocationData[_address].push(_releasePeriodInSeconds);
        emit Lock(_address, _amount); // 成都链安 // 触发 Lock 事件
    }
    // 成都链安 // 获取指定地址在指定时间_time 可用代币数量
    function getUnlockedBalance(
        address _address,
        uint256 _time,
        uint256 _holderBalance
    )
        public
        view
        returns (uint256)
    {
        uint256 blockedAmount = manuallyLockedBalances[_address]; // 成都链安 // 声明变量 blockedAmount，记录指定地址_address 被锁定的代币数量，初始为手动锁仓的代币数量
        // 成都链安 // 如果指定地址_address 未进行分配锁仓，则返回其代币余额与手动锁仓代币数量的差
        if (lockedAllocationData[_address].length == 0) {
            return _holderBalance.sub(blockedAmount);
        }
        uint256[] memory  addressLockupData = lockedAllocationData[_address]; // 成都链安 // 声明数组 addressLockupData，存储锁仓的基本信息
        // 成都链安 // 遍历分配锁仓信息
        for (uint256 i = 0; i < addressLockupData.length / 4; i++) {
```

```
            uint256 lockedAt = addressLockupData[i.mul(4)]; // 成都链安 // 获取第 i 个锁仓的开始时间
            uint256 lockedBalance = addressLockupData[i.mul(4).add(1)]; // 成都链安 // 获取第 i 个锁仓的代币数量
            uint256 lockPeriodInSeconds = addressLockupData[i.mul(4).add(2)]; // 成都链安 // 获取第 i 个锁仓的锁仓总时长
            uint256 _releasePeriodInSeconds = addressLockupData[
                i.mul(4).add(3) // 成都链安 // 获取第 i 个锁仓的释放周期的时长
            ];
            if (lockedAt == 0 && true == isPostponedStart) { // 成都链安 // 如果锁仓的开始时间戳为零且开启锁仓延迟
                if (postponedStartDate == 0) {
                    blockedAmount = blockedAmount.add(lockedBalance); // 成都链安 // 如果锁仓延迟的开始时间戳为零，则第 i 个锁仓的锁定代币数量为锁仓分配全部数量；注意，该操作逻辑为项目方原本意图，忽略影响
                    continue;
                }
                lockedAt = postponedStartDate; // 成都链安 // 设置第 i 个锁仓的开始时间为 postponedStartDate
            }
            if (lockedAt > _time) { // 成都链安 // 如果时间_time 小于第 i 个锁仓开始时间，则第 i 个锁仓的锁定代币数量为锁仓分配全部数量
                blockedAmount = blockedAmount.add(lockedBalance);
                continue;
            }
            if (lockedAt.add(lockPeriodInSeconds) > _time) { // 成都链安 // 如果第 i 个锁仓的开始时间与锁仓总时间的和大于时间_time，则进入该 if 分支
                if (lockedBalance == 0) { // 成都链安 // 如果第 i 个锁仓的锁仓代币数量为 0
                    blockedAmount = _holderBalance; // 成都链安 // 设置 blockedAmount 为对应地址的代币余额，仍锁定代币数量为指定地址的代币余额；注意，该操作逻辑为项目方原本意图，忽略影响
                    break;
                } else { // 成都链安 // 否则进入该分支
                    uint256 tokensUnlocked;
                    if (_releasePeriodInSeconds > 0) { // 成都链安 // 如果释放周期时长大于 0，则计算该地址在指定时间_time 释放的代币数量
                        uint256 duration = (_time.sub(lockedAt))
                            .div(_releasePeriodInSeconds); // 成都链安 // 声明局部变量 duration，获取在指定时间_time 已经过时间间隔数量
                        tokensUnlocked = lockedBalance.mul(duration)
                            .mul(_releasePeriodInSeconds)
                            .div(lockPeriodInSeconds); // 成都链安 // 获取在指定时间_time 已释放代币数量
                    }
                    blockedAmount = blockedAmount
                        .add(lockedBalance)
                        .sub(tokensUnlocked); // 成都链安 // 更新该地址的仍被锁仓的代币总量
                }
```

```
            }
        }

        return _holderBalance.sub(blockedAmount); // 成都链安 // 获取可用代币总量
    }
    // 成都链安 // 设置手动锁仓函数，调用者为指定地址_holder 设置手动锁仓代币数量
    function setManuallyLockedForAddress (
        address _holder,
        uint256 _balance
    )
        public
        requirePermission(CAN_LOCK_TOKENS) // 成都链安 // 要求调用者具有设置锁仓的权限
    {
        manuallyLockedBalances[_holder] = _balance; // 成都链安 // 设置指定地址_holder 的手动锁仓代币数量
    }
    // 成都链安 // 设置锁仓延迟开始时间函数
    function setPostponedStartDate(uint256 _postponedStartDate)
        public
        requirePermission(CAN_LOCK_TOKENS) // 成都链安 // 要求调用者具有设置锁仓的权限
    {
        postponedStartDate = _postponedStartDate; // 成都链安 // 设置锁仓延迟开始时间为_postponedStartDate

    }
}


// 成都链安 // 文件名称：contracts/managment/Management.sol
pragma solidity 0.5.17;
// 成都链安 // 导入合约所需的其他合约模块
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "./Constants.sol";


contract Management is Ownable, Constants {

    // Contract Registry
    mapping (uint256 => address payable) public contractRegistry; // 成都链安 // 声明 mapping 变量 contractRegistry，存储注册合约对应的地址

    // Permissions
    mapping (address => mapping(uint256 => bool)) public permissions; // 成都链安 // 声明 mapping 变量 permissions，存储指定地址对应的权限

    event PermissionsSet(
        address subject,
        uint256 permission,
```

```
        bool value
    ); // 成都链安 // 声明权限设置事件

    event ContractRegistered(
        uint256 key,
        address source,
        address target
    ); // 成都链安 // 声明合约注册事件
    // 成都链安 // 设置权限函数，合约所有者设置指定地址的权限
    function setPermission(
        address _address,
        uint256 _permission,
        bool _value
    )
        public
        onlyOwner
    {
        permissions[_address][_permission] = _value;
        emit PermissionsSet(_address, _permission, _value); // 成都链安 // 触发
PermissionsSet 事件
    }
    // 成都链安 // 注册合约函数，合约所有者注册合约
    function registerContract(
        uint256 _key,
        address payable _target
    )
        public
        onlyOwner
    {
        contractRegistry[_key] = _target; // 成都链安 // 注册合约
        emit ContractRegistered(_key, address(0), _target); // 成都链安 // 触发
ContractRegistered 事件
    }
    // 成都链安 // 设置指定地址的白名单状态
    function setWhitelisted(
        address _address,
        bool _value
    )
        public
    {
        require(
            permissions[msg.sender][CAN_SET_WHITELISTED] == true,
            ERROR_ACCESS_DENIED
        ); // 成都链安 // 要求调用者具有添加白名单的权限

        permissions[_address][WHITELISTED] = _value; // 成都链安 // 设置指定地址_address 的白
名单状态为_value
```

```
        emit PermissionsSet(_address, WHITELISTED, _value); // 成都链安 // 触发 PermissionsSet 事件
    }

}


// 成都链安 // 文件名称：contracts/managment/Managed.sol
pragma solidity 0.5.17;
// 成都链安 // 导入合约所需的其他合约模块
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "./Constants.sol";
import "./Management.sol";


contract Managed is Ownable, Constants {

    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

    Management public management;
    // 成都链安 // 修饰器，要求调用者必须具有相应的权限
    modifier requirePermission(uint256 _permissionBit) {
        require(
            hasPermission(msg.sender, _permissionBit),
            ERROR_ACCESS_DENIED
        );
        _;
    }
    // 成都链安 // 修饰器，要求调用者必须是注册合约
    modifier canCallOnlyRegisteredContract(uint256 _key) {
        require(
            msg.sender == management.contractRegistry(_key),
            ERROR_ACCESS_DENIED
        );
        _;
    }
    // 成都链安 // 修饰器，要求_key 对应的注册合约地址不为 0 地址
    modifier requireContractExistsInRegistry(uint256 _key) {
        require(
            management.contractRegistry(_key) != address(0),
            ERROR_NO_CONTRACT
        );
        _;
    }
    // 成都链安 // 构造函数，初始化 Management 合约实例
    constructor(address _managementAddress) public {
```

```
        management = Management(_managementAddress);
    }
    // 成都链安 // 设置管理合约函数，合约所有者调用该函数设置管理合约
    function setManagementContract(address _management) public onlyOwner {
        require(address(0) != _management, ERROR_NO_CONTRACT); // 成都链安 // _management 非零地址检查

        management = Management(_management); // 成都链安 // 设置 Management 合约实例
    }
    // 成都链安 // 权限查询函数，查询指定地址_subject 是否具有指定权限
    function hasPermission(address _subject, uint256 _permissionBit)
        internal
        view
        returns (bool)
    {
        return management.permissions(_subject, _permissionBit);
    }

}


// 成都链安 // 文件名称：contracts/managment/Constants.sol
pragma solidity 0.5.17;

// 成都链安 // 声明主合约所需常量
contract Constants {
    // Permissions bit constants
    uint256 public constant CAN_MINT_TOKENS = 0;
    uint256 public constant CAN_BURN_TOKENS = 1;
    uint256 public constant CAN_UPDATE_STATE = 2;
    uint256 public constant CAN_LOCK_TOKENS = 3;
    uint256 public constant CAN_UPDATE_PRICE = 4;
    uint256 public constant CAN_INTERACT_WITH_ALLOCATOR = 5;
    uint256 public constant CAN_SET_ALLOCATOR_MAX_SUPPLY = 6;
    uint256 public constant CAN_PAUSE_TOKENS = 7;
    uint256 public constant ECLIUDED_ADDRESSES = 8;
    uint256 public constant WHITELISTED = 9;
    uint256 public constant SIGNERS = 10;
    uint256 public constant EXTERNAL_CONTRIBUTORS = 11;
    uint256 public constant CAN_SEE_BALANCE = 12;
    uint256 public constant CAN_CANCEL_TRANSACTION = 13;
    uint256 public constant CAN_ALLOCATE_REFERRAL_TOKENS = 14;
    uint256 public constant CAN_SET_REFERRAL_MAX_SUPPLY = 15;
    uint256 public constant MANUAL_TOKENS_ALLOCATION = 16;
    uint256 public constant CAN_SET_WHITELISTED = 17;

    // Contract Registry keys
    uint256 public constant CONTRACT_TOKEN = 1;
```

```
    uint256 public constant CONTRACT_PRICING = 2;
    uint256 public constant CONTRACT_CROWDSALE = 3;
    uint256 public constant CONTRACT_ALLOCATOR = 4;
    uint256 public constant CONTRACT_AGENT = 5;
    uint256 public constant CONTRACT_FORWARDER = 6;
    uint256 public constant CONTRACT_REFERRAL = 7;
    uint256 public constant CONTRACT_STATS = 8;
    uint256 public constant CONTRACT_LOCKUP = 9;

    uint256 public constant YEAR_IN_SECONDS = 31556952;
    uint256 public constant SIX_MONTHS =  15778476;
    uint256 public constant MONTH_IN_SECONDS = 2629746;

    string public constant ERROR_ACCESS_DENIED = "ERROR_ACCESS_DENIED";
    string public constant ERROR_WRONG_AMOUNT = "ERROR_WRONG_AMOUNT";
    string public constant ERROR_NO_CONTRACT = "ERROR_NO_CONTRACT";
    string public constant ERROR_NOT_AVAILABLE = "ERROR_NOT_AVAILABLE";
}
```

成都链安
BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号