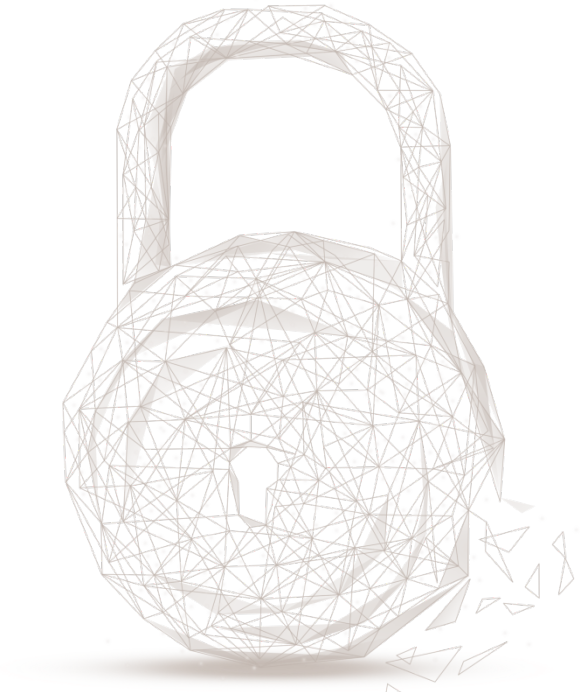
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202103192327

报告查询名称：wata

审计合约名称	审计合约地址	审计合约链接
MLTC	部署后填写	部署后填写
Staking	部署后填写	部署后填写
Vault	部署后填写	部署后填写
LpStaking	部署后填写	部署后填写

合约 GitHub 地址：

https://github.com/easy2mine/wata.finance/commit/9b07a40b0f0abb5a962b4e74d27f9ea7931d5438

commit hash:

9b07a40b0f0abb5a962b4e74d27f9ea7931d5438

合约审计开始日期：2021.03.17

合约审计完成日期：2021.03.19

审计结果：通过

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	编译器版本安全审计	通过
		弃用项审计	通过
		冗余代码审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
2	通用漏洞审计	整型溢出审计	通过
		重入攻击审计	通过
		伪随机数生成审计	通过
		交易顺序依赖审计	通过

		拒绝服务攻击审计	通过
		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
		tx.origin 使用安全审计	通过
		重放攻击审计	通过
		变量覆盖审计	通过
3	业务审计	业务逻辑审计	通过
		业务实现审计	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wata项目智能合约代码规范性，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，wata项目智能合约通过所有检测项，合约审计结果为通过。**以下为本合约详细审计信息。

一. 代码规范审计

1. 编译器版本安全审计

老版本的编译器可能会导致各种已知的安全问题，建议开发者在代码中指定合约代码采用最新的编译器版本，并消除编译器告警。

- **安全建议：无**
- **审计结果：通过**

2. 弃用项审计

Solidity智能合约开发语言处于快速迭代中，部分关键字已被新版本的编译器弃用，如throw，years等，为了消除其可能导致的隐患，合约开发者不应该使用当前编译器版本已弃用的关键字。

- **安全建议：无**
- **审计结果：通过**

3. 冗余代码审计

智能合约中的冗余代码会降低代码可读性，并可能需要消耗更多的gas用于合约部署，建议消除冗余代码。

- **安全建议：无**
- **审计结果：通过**

4. require/assert 使用审计

Solidity使用状态恢复异常来处理错误。这种机制将会撤消对当前调用(及其所有子调用)中的状态所做的所有更改，并向调用者标记错误。函数assert和require可用于检查条件并在条件不满足时抛出异常。assert函数只能用于测试内部错误，并检查非变量。require函数用于确认条件有效性，例如输入变量，或合约状态变量是否满足条件，或验证外部合约调用的返回值。

- **安全建议：无**
- **审计结果：通过**

5. gas 消耗审计

Heco虚拟机执行合约代码需要消耗gas，当gas不足时，代码执行会抛出out of gas异常，并撤销所有状态变更。合约开发者需要控制代码的gas消耗，避免因为gas不足导致函数执行一直失败。

- **安全建议：无**
- **审计结果：通过**

二. 通用漏洞审计

1. 整型溢出审计

整型溢出是很多语言都存在的安全问题，它们在智能合约中尤其危险。Solidity最多能处理256位的数字(2**256-1)，最大数字增加1会溢出得到0。同样，当数字为uint类型时，0减去1会下溢得到最大数字值。溢出情况会导致不正确的结果，特别是如果其可能的结果未被预期，可能会影响程序的可靠性和安全性。

- **安全建议：无**
- **审计结果：通过**

2. 重入攻击审计

重入漏洞是最典型的智能合约漏洞，该漏洞原因是Solidity中的call.value()函数在被用来发送HT的时候会消耗它接收到的所有gas，当调用call.value()函数发送HT的逻辑顺序存在错误时，就会存在重入攻击的风险。

- **安全建议：无**
- **审计结果：通过**

3. 伪随机数生成审计

智能合约中可能会使用到随机数，在solidity下常见的是用block区块信息作为随机因子生成，但是这样使用是不安全的，区块信息是可以被矿工控制或被攻击者在交易时获取到，这类随机数在一定程度上是可预测或可碰撞的，比较典型的例子就是fomo3d的airdrop随机数可以被碰撞。

- **安全建议：无**
- **审计结果：通过**

4. 交易顺序依赖审计

在Heco的交易打包执行过程中，面对相同难度的交易时，矿工往往会选择gas费用高的优先打包，因此用户可以指定更高的gas费用，使自己的交易优先被打包执行。

- **安全建议：无**
- **审计结果：通过**

5. 拒绝服务攻击审计

拒绝服务攻击，即Denial of Service，可以使目标无法提供正常的服务。在Heco智能合约中也会存在此类问题，由于智能合约的不可更改性，该类攻击可能使得合约永远无法恢复正常工作状态。导致智能合约拒绝服务的原因有很多种，包括在作为交易接收方时的恶意revert，代码设计缺陷导致gas耗尽等等。

- **安全建议：无**
- **审计结果：通过**

6. 函数调用权限审计

智能合约如果存在高权限功能，如：铸币，自毁，change owner等，需要对函数调用做权限限制，避免权限泄露导致的安全问题。

- **安全建议：无**
- **审计结果：通过**

7. call/delegatecall安全审计

Solidity中提供了call/delegatecall函数来进行函数调用，如果使用不当，会造成call注入漏洞，例如call的参数如果可控，则可以控制本合约进行越权操作或调用其他合约的危险函数。

- **安全建议：无**
- **审计结果：通过**

8. 返回值安全审计

在Solidity中存在transfer()，send()，call.value()等方法中，transfer转账失败交易会回滚，而send和call.value转账失败会return false，如果未对返回做正确判断，则可能会执行到未预期的逻辑；另外在TRC20 Token的transfer/transferFrom功能实现中，也要避免转账失败return false的情况，以免造成假充值漏洞。

- **安全建议：无**
- **审计结果：通过**

9. tx.origin使用安全审计

在Heco智能合约的复杂调用中，tx.origin表示交易的初始创建者地址，如果使用tx.origin进行权限判断，可能会出现错误；另外，如果合约需要判断调用方是否为合约地址时则需要使用tx.origin，不能使用extcodesize。

- **安全建议：无**
- **审计结果：通过**

10. 重放攻击审计

重放攻击是指如果两份合约使用了相同的代码实现，并且身份鉴权在传参中，当用户在向一份合约中执行一笔交易，交易信息可以被复制并且向另一份合约重放执行该笔交易。

- **安全建议：无**
- **审计结果：通过**

11. 变量覆盖审计

Heco存在着复杂的变量类型，例如结构体，动态数组等，如果使用不当，对其赋值后，可能导致覆盖已有状态变量的值，造成合约执行逻辑异常。

- **安全建议：无**
- **审计结果：通过**

三. 业务审计

3.1 MLTC代币合约审计

(1) mLTC代币基本信息

代币名称	Litecoin Standard Hashrate Token
代币简称	mLTC
代币精度	18
代币总量	初始总量为0，不可销毁，可铸币，铸币无上限
代币类型	HRC-20

表 1 代币基本信息

(2) HRC-20 代币标准函数

- **业务描述：**MLTC 合约所实现的是一个标准的 HRC-20 代币，其相关函数符合 HRC-20 代币标准规范。需要注意的是，用户可以使用 approve 函数设置对指定地址的授权值，但为了避免多重授权，建议在需要修改授权值时，不要直接使用 approve 函数进行修改，而是使用 increaseAllowance 和 decreaseAllowance 函数对当前授权值进行增加和减少。
- **相关函数：**name、symbol、decimals、totalSupply、balanceOf、llowance、transfer, transferFrom、approve、increaseAllowance、decreaseAllowance
- **安全建议：无**
- **审计结果：通过**

(3) mint 函数

- **业务描述：**如下图所示，合约的 owner 可调用此函数向指定地址铸币，并且铸币无上限。

```
function mint(address to, uint256 value) external onlyOwner {
    _mint(to, value);
}
```

图 1 mint 函数源码

- **相关函数：**mint、mint、beforeTokenTransfer
- **安全建议：无**
- **审计结果：通过**

(4) permit 函数

- **业务描述：**如下图所示，合约中的 permit 函数，函数调用者可以使用签名认证进行授权操作。

```
function permit(
  address _owner,
  address spender,
  uint256 value,
  uint256 deadline,
  uint8 v,
  bytes32 r,
  bytes32 s
) external {
  require(deadline >= block.timestamp, 'MLTC: EXPIRED');
  bytes32 digest =
    keccak256(
      abi.encodePacked(
        '\x19\x01',
        DOMAIN_SEPARATOR,
        keccak256(abi.encode(PERMIT_TYPEHASH, _owner, spender, value, nonces[_owner]++, deadline))
      )
    );
  address recoveredAddress = ecrecover(digest, v, r, s);
  require(
    recoveredAddress != address(0) && recoveredAddress == _owner,
    'MLTC: INVALID_SIGNATURE'
  );
  _approve(_owner, spender, value);
}
```

图 2 permit 函数源码

- **相关函数：**permit
- **安全建议：无**
- **审计结果：通过**

(5) 相关治理函数

- **业务描述：**合约实现了 pause，unpause 函数用于合约交易暂停和交易开始。pause 函数由 owner 调用，暂停合约交易功能。unpause 函数由 owner 调用，开启合约交易功能。

```
function pause() external onlyOwner {
  _pause();
}

function unpause() external onlyOwner {
  _unpause();
}
```

图 3 pause，unpause 函数源码

- **相关函数：**pause、_pause、pause、unpause
- **安全建议：无**
- **审计结果：通过**

3.2 Vault 合约审计

(1) 抵押初始化

- **业务描述：**合约的“抵押-奖励”模式需要初始化相关参数（设置管理员owner地址，设置奖励代币hltc，hDOGE和hPT的地址，设置抵押池_mLTCStaking和_mLTCUSDTLpStaking地址），通过修饰器initializer限制初始化次数。

```
function initialize(
  address _owner,
  address _hLTC,
  address _hDOGE,
  address _hPT,
  address _mLTCStaking,
  address _mLTCUSDTLpStaking
) external virtual initializer {
  require(_owner != address(0), 'new owner is the zero address');
  owner = _owner;
  hLTC = _hLTC;
  hDOGE = _hDOGE;
  hPT = _hPT;
  mLTCStaking = _mLTCStaking;
  mLTCUSDTLpStaking = _mLTCUSDTLpStaking;
}
```

图 4 initialize 函数源码截图

```
modifier initializer() {
  uint256 revision = getRevision();

  require(
    initializing || isConstructor() || revision > lastInitializedRevision,
    'Contract instance has already been initialized'
  );

  bool isTopLevelCall = !initializing;
  if (isTopLevelCall) {
    initializing = true;
    lastInitializedRevision = revision;
  }

  _;

  if (isTopLevelCall) {
    initializing = false;
  }
}
```

图 5 initializer 修饰器源码截图

- **相关函数：**initialize
- **安全建议：无**
- **审计结果：通过**

(2) 设置周期奖励

- **业务描述：**合约实现了addNewPeriodReward函数用于设置周期奖励，该函数用户预先授权该合约地址，通过调用合约中的safeTransferFrom函数向本合约地址转入指定数量的奖励代币，同时设置mLTCStaking和mLTCUSDTLpStaking抵押池的周期奖励数据，并更新奖励代币总量，周期开始时间和周期结束时间。该函数仅由管理员调用，同时满足周期结束时间大于当前时间，并且由管理员设置的周期结束时间必须大于等于上个周期结束时间。

```
function addNewPeriodReward(
  uint256 _rewardHLTC,
  uint256 _rewardHDOGE,
  uint256 _rewardHPT,
  uint256 _endTime
) public onlyOwner {
  require(_endTime > block.timestamp, '_endTime >= block.timestamp !');
  require(
    _endTime >= curPeriodEnd,
    'The endTime of the latest period must be greater than the end time of the current period'
  );

  curPeriodStart = block.timestamp;
  curPeriodEnd = _endTime;

  if (_rewardHLTC > 0) {
    IERC20(hLTC).safeTransferFrom(msg.sender, address(this), _rewardHLTC);
  }
  if (_rewardHDOGE > 0) {
    IERC20(hDOGE).safeTransferFrom(msg.sender, address(this), _rewardHDOGE);
  }
  if (_rewardHPT > 0) {
    IERC20(hPT).safeTransferFrom(msg.sender, address(this), _rewardHPT);
  }
  IStaking(mLTCStaking).addNewPeriodReward(_rewardHLTC, _rewardHDOGE, _rewardHPT, _endTime);
  ILpStaking(mLTCUSDTLpStaking).addNewPeriodReward(
    _rewardHLTC,
    _rewardHDOGE,
    _rewardHPT,
    _endTime
  );
  totalRewardHLTC = totalRewardHLTC.add(_rewardHLTC);
  totalRewardHDOGE = totalRewardHDOGE.add(_rewardHDOGE);
  totalRewardHPT = totalRewardHPT.add(_rewardHPT);
  emit AddNewPeriodReward(_rewardHLTC, _rewardHDOGE, _rewardHPT, curPeriodStart, curPeriodEnd);
}
```

图 6 addNewPeriodReward 函数源码截图

- **相关函数：**addNewPeriodReward、safeTransferFrom
- **安全建议：无**
- **审计结果：通过**

(3) 奖励代币转账函数

- **业务描述：**合约实现了claimRewardHLTC，claimRewardHDOGE和claimRewardHPT函数用于奖励代币转账，限制函数调用者只能为mLTCStaking和mLTCUSDTLpStaking抵押池地址。

```
function claimRewardHLTC(address to, uint256 amount) external {
  require(to != address(0), 'to is the zero address');
  require(msg.sender == mLTCStaking || msg.sender == mLTCUSDTLpStaking, 'No permissions');
  IERC20(hLTC).safeTransfer(to, amount);
}

function claimRewardHDOGE(address to, uint256 amount) external {
  require(to != address(0), 'to is the zero address');
  require(msg.sender == mLTCStaking || msg.sender == mLTCUSDTLpStaking, 'No permissions');
  IERC20(hDOGE).safeTransfer(to, amount);
}

function claimRewardHPT(address to, uint256 amount) external {
  require(to != address(0), 'to is the zero address');
  require(msg.sender == mLTCStaking || msg.sender == mLTCUSDTLpStaking, 'No permissions');
  IERC20(hPT).safeTransfer(to, amount);
}
```

图 7 claimRewardHLTC&claimRewardHDOGE&claimRewardHPT 函数源码截图

- **相关函数：**claimRewardHLTC、claimRewardHDOGE、claimRewardHPT、safeTransfer
- **安全建议：无**
- **审计结果：通过**

(4) 其它相关函数

- **业务描述：**transferOwnership函数仅由管理员调用，用于更新owner地址。

```
function transferOwnership(address newOwner) external onlyOwner {
  require(newOwner != address(0), 'new owner is the zero address');
  emit OwnershipTransferred(owner, newOwner);
  owner = newOwner;
}
```

图 8 其他相关函数源码截图

- **相关函数：**transferOwnership
- **安全建议：无**
- **审计结果：通过**

3.3 Staking 合约审计

Staking 合约和 LpStaking 合约，除了奖励计算，地址，名称外整体逻辑是相同的，下面截图基于 Staking 合约。

(1) 抵押初始化

- **业务描述：**合约的“抵押-奖励”模式需要初始化相关参数（设置管理员owner地址，设置抵押代币mLTC地址，设置抵押池mLTCUSDTLpStaking地址，设置vault合约地址），通过修饰器initializer限制初始化次数。

```
function initialize(
  address _owner,
  address _mLTC,
  address _mLTCUSDTLpStaking,
  address _vault
) external virtual initializer {
  require(_owner != address(0), 'new owner is the zero address');
  super.initialize();
  owner = _owner;
  mLTC = _mLTC;
  mLTCUSDTLpStaking = _mLTCUSDTLpStaking;
  vault = _vault;
}
```

图 9 initialize 函数源码截图

```
modifier initializer() {
  uint256 revision = getRevision();

  require(
    initializing || isConstructor() || revision > lastInitializedRevision,
    'Contract instance has already been initialized'
  );

  bool isTopLevelCall = !initializing;
  if (isTopLevelCall) {
    initializing = true;
    lastInitializedRevision = revision;
  }

  _;

  if (isTopLevelCall) {
    initializing = false;
  }
}
```

图 10 initializer 修饰器源码截图

- **相关函数：**initialize
- **安全建议：无**
- **审计结果：通过**

(2) 设置周期奖励

- **业务描述：**合约实现了addNewPeriodReward函数用于设置周期奖励，该函数只能通过valut合约地址调用，通过判断当前时间是否在上个周期内，如果当前时间在上个周期范围内，则计算上个

周期剩余的奖励加入到这个周期内，同时更新当前奖励代币数量，周期开始时间，周期结束时间。每次调用该函数设置周期奖励时，通过修饰器refreshAccountReward更新账户奖励，并更新rewardLastUpdateTime上次更新时间。

```
function addNewPeriodReward(
  uint256 rewardHLTC,
  uint256 rewardHDOGE,
  uint256 rewardHPT,
  uint256 _endTime
) external onlyVault refreshAccountReward(address(0)) {
  uint256 _startTime = block.timestamp;
  // Check whether the current period has finished, if not, the remaining reward will be put on the next peroid.
  if (curPeriodStart < _startTime && curPeriodEnd > _startTime) {
    uint256 factor = 100000000;
    uint256 left = curPeriodEnd.sub(_startTime).mul(factor).div(curPeriodEnd.sub(curPeriodStart));
    curPeriodHLTC = curPeriodHLTC.mul(left).div(factor).add(rewardHLTC);
    curPeriodHDOGE = curPeriodHDOGE.mul(left).div(factor).add(rewardHDOGE);
    curPeriodHPT = curPeriodHPT.mul(left).div(factor).add(rewardHPT);
  } else {
    // The current peroid has already been finished.
    curPeriodHLTC = rewardHLTC;
    curPeriodHDOGE = rewardHDOGE;
    curPeriodHPT = rewardHPT;
  }

  curPeriodStart = _startTime;
  curPeriodEnd = _endTime;
}
```

图 11 addNewPeriodReward 函数源码截图

```
  modifier refreshAccountReward(address account) {
    if (block.timestamp > startMiningTime) {
      sumHLTCPerToken = sumHLTCPerToken.add(calRewardPerToken(curPeriodHLTC));
      sumHDOGEPerToken = sumHDOGEPerToken.add(calRewardPerToken(curPeriodHDOGE));
      sumHPTPerToken = sumHPTPerToken.add(calRewardPerToken(curPeriodHPT));
      rewardLastUpdateTime = block.timestamp;
      if (account != address(0)) {
        userUnclaimedHLTC[account] = getUnclaimedHLTC(account);
        userSumHLTCPerToken[account] = sumHLTCPerToken;

        userUnclaimedHDOGE[account] = getUnclaimedHDOGE(account);
        userSumHDOGEPerToken[account] = sumHDOGEPerToken;

        userUnclaimedHPT[account] = getUnclaimedHPT(account);
        userSumHPTPerToken[account] = sumHPTPerToken;
      }
    }
    _;
  }
```

图 12 refreshAccountReward 修饰器函数源码截图

```
function calRewardPerToken(uint256 _curRewardAmount) internal view returns (uint256) {
  if (block.timestamp <= startMiningTime || totalSupply == 0) {
    return 0;
  }
  uint256 startTime = rewardLastUpdateTime;
  uint256 endTime = block.timestamp;
  if (startTime == 0) {
    startTime = startMiningTime;
  }
  if (startTime < curPeriodStart) {
    startTime = curPeriodStart;
  }
  if (endTime > curPeriodEnd) {
    endTime = curPeriodEnd;
  }
  if (startTime < endTime && startTime >= curPeriodStart && endTime <= curPeriodEnd) {
    uint256 mLTCTotalSupply = IERC20(mLTC).totalSupply();
    uint256 curRewardAmount = _curRewardAmount.mul(totalSupply).div(mLTCTotalSupply);

    return
      curRewardAmount.mul(1e18).mul(endTime.sub(startTime)).div(totalSupply).div(
        curPeriodEnd.sub(curPeriodStart)
      );
  }
  return 0;
}
```

图 13 calRewardPerToken 函数源码截图

- **相关函数：** addNewPeriodReward、calRewardPerToken、getUnclaimedHLTC、getUnclaimedHDOGE、getUnclaimedHPT、totalSupply
- **安全建议：无**
- **审计结果：通过**

(3) 抵押代币

- **业务描述：**合约实现了stake函数用于抵押代币，用户预先授权该合约地址，通过调用合约中的safeTransferFrom函数，向抵押池地址转入代币，同时更新账户余额和抵押池代币总量。每次调用该函数抵押代币时通过修饰器refreshAccountReward更新账户奖励数据，同时调用修饰器nonReentrant防止重入，并调用修饰器notifyLpStaking在LpStaking合约中刷新奖励。

```
function stake(uint256 amount)
  external
  nonReentrant
  refreshAccountReward(msg.sender)
  notifyLpStaking()
{
  require(amount > 0, 'Cannot stake 0');
  mLTCBalances[msg.sender] = mLTCBalances[msg.sender].add(amount);
  totalSupply = totalSupply.add(amount);
  IERC20(mLTC).safeTransferFrom(msg.sender, address(this), amount);
  emit Staked(msg.sender, amount);
}
```

图 14 withdraw 函数源码截图

```
modifier notifyLpStaking() {
  ILpStaking(mLTCUSDTLpStaking).refreshReward();
  _;
}
```

图 15 notifyLpStaking 修饰器源码截图

```
function refreshReward() external refreshAccountReward(address(0)) {}
```

图 16 refreshReward 函数源码截图（LpStaking 合约）

- **相关函数：** stake、calRewardPerToken、getUnclaimedHLTC、getUnclaimedHDOGE，getUnclaimedHPT、refreshReward
- **安全建议：无**
- **审计结果：通过**

(4) 提取抵押代币

- **业务描述：**合约实现了withdraw函数用于提取已抵押的代币，通过调用合约中的safeTransfer函数，合约地址将指定数量的代币转至函数调用者（用户）地址，同时更新账户余额和抵押池代币总量。每次调用该函数抵押代币时通过修饰器refreshAccountReward更新账户奖励数据，同时调用修饰器nonReentrant防止重入，并调用修饰器notifyLpStaking在ILpStaking合约中刷新奖励。

```
function withdraw(uint256 amount)
  public
  nonReentrant
  refreshAccountReward(msg.sender)
  notifyLpStaking()
{
  require(amount > 0, 'Cannot withdraw 0');
  totalSupply = totalSupply.sub(amount);
  mLTCBalances[msg.sender] = mLTCBalances[msg.sender].sub(amount);
  IERC20(mLTC).safeTransfer(msg.sender, amount);
  emit Withdrawn(msg.sender, amount);
}
```

图 17 withdraw 函数源码截图

- **相关函数：** withdraw、calRewardPerToken、getUnclaimedHLTC、getUnclaimedHDOGE、getUnclaimedHPT、refreshReward
- **安全建议：无**
- **审计结果：通过**

(5) 领取抵押奖励

- **业务描述：**合约实现了claimReward函数用于领取抵押奖励，通过调用vault合约中的safeTransfer函数，合约地址将指定数量（用户的全部抵押奖励）的代币转至函数调用者（用户）地址，每次调用该函数抵押代币时通过修饰器refreshAccountReward更新账户奖励数据，同时调用修饰器nonReentrant防止重入，并调用修饰器notifyLpStaking在ILpStaking合约中刷新奖励。修饰器nonEmergencyStop由管理员控制用户取出奖励代币。

```
function claimReward() public nonReentrant nonEmergencyStop refreshAccountReward(msg.sender) {
  uint256 hltc = userUnclaimedHLTC[msg.sender];
  if (hltc > 0) {
    userUnclaimedHLTC[msg.sender] = 0;
    IVault(vault).claimRewardHLTC(msg.sender, hltc);
  }
  uint256 hdoge = userUnclaimedHDOGE[msg.sender];
  if (hdoge > 0) {
    userUnclaimedHDOGE[msg.sender] = 0;
    IVault(vault).claimRewardHDOGE(msg.sender, hdoge);
  }
  uint256 hpt = userUnclaimedHPT[msg.sender];
  if (hpt > 0) {
    userUnclaimedHPT[msg.sender] = 0;
    IVault(vault).claimRewardHPT(msg.sender, hpt);
  }
  emit RewardPaid(msg.sender, hltc, hdoge, hpt);
}
```

图 18 claimReward 函数源码截图

```
function claimRewardHLTC(address to, uint256 amount) external {
  require(to != address(0), 'to is the zero address');
  require(msg.sender == mLTCStaking || msg.sender == mLTCUSDTLpStaking, 'No permissions');
  IERC20(hLTC).safeTransfer(to, amount);
}

function claimRewardHDOGE(address to, uint256 amount) external {
  require(to != address(0), 'to is the zero address');
  require(msg.sender == mLTCStaking || msg.sender == mLTCUSDTLpStaking, 'No permissions');
  IERC20(hDOGE).safeTransfer(to, amount);
}

function claimRewardHPT(address to, uint256 amount) external {
  require(to != address(0), 'to is the zero address');
  require(msg.sender == mLTCStaking || msg.sender == mLTCUSDTLpStaking, 'No permissions');
  IERC20(hPT).safeTransfer(to, amount);
}
```

图 19 claimRewardHLTC&claimRewardHDOGE&claimRewardHPT 函数源码截图（Vault 合约）

```
}

modifier nonEmergencyStop() {
  require(emergencyStop == false, 'emergency stop');
  _;
}
```

图 20 nonEmergencyStop 修饰器源码截图

- **相关函数：** claimReward、calRewardPerToken、getUnclaimedHLTC、getUnclaimedHDOGE，getUnclaimedHPT、refreshReward、claimRewardHLTC、claimRewardHDOGE、claimRewardHPT，safeTransfer
- **安全建议：无**
- **审计结果：通过**

(6) 提取所有代币并领取奖励

- **业务描述：**合约实现了exit函数用于调用者提取所有代币并领取奖励，调用withdraw函数提取全部已抵押的代币，调用claimReward函数领取完调用者的抵押奖励，结束“抵押-奖励”模式参与。此时用户地址由于其已抵押代币数量为空，无法获得新的抵押奖励。

```
function exit() external {
  withdraw(mLTCBalances[msg.sender]);
  claimReward();
}
```

图 21 exit 函数源码截图

- **相关函数：**exit、withdraw、claimReward
- **安全建议：无**
- **审计结果：通过**

(7) 其他相关函数功能介绍

- **业务描述：**合约用户可通过调用getUnclaimedHLTC，getUnclaimedHDOGE和getUnclaimedHPT函数查询当前用户获得的奖励；由owner调用transferOwnership函数可设置新的管理员；由owner调用setEmergencyStop函数可控制用户领取奖励。由owner调用setStartMiningTime函数用于设置开始奖励计算时间。

```
function getUnclaimedHLTC(address account) public view returns (uint256) {
  uint256 rewardPerToken =
    sumHLTCPerToken.add(calRewardPerToken(curPeriodHLTC)).sub(userSumHLTCPerToken[account]);
  return mLTCBalances[account].mul(rewardPerToken).div(1e18).add(userUnclaimedHLTC[account]);
}

function getUnclaimedHDOGE(address account) public view returns (uint256) {
  uint256 rewardPerToken =
    sumHDOGEPerToken.add(calRewardPerToken(curPeriodHDOGE)).sub(userSumHDOGEPerToken[account]);
  return mLTCBalances[account].mul(rewardPerToken).div(1e18).add(userUnclaimedHDOGE[account]);
}

function getUnclaimedHPT(address account) public view returns (uint256) {
  uint256 rewardPerToken =
    sumHPTPerToken.add(calRewardPerToken(curPeriodHPT)).sub(userSumHPTPerToken[account]);
  return mLTCBalances[account].mul(rewardPerToken).div(1e18).add(userUnclaimedHPT[account]);
}
```

图 22 相关函数源码（一）

```
function transferOwnership(address newOwner) external onlyOwner {
  require(newOwner != address(0), 'new owner is the zero address');
  emit OwnershipTransferred(owner, newOwner);
  owner = newOwner;
}

function setEmergencyStop(bool _emergencyStop) external onlyOwner {
  emergencyStop = _emergencyStop;
}

function setStartMiningTime(uint256 _startMiningTime) external onlyOwner {
  require(_startMiningTime > block.timestamp, 'nonlegal startMiningTime.');
  startMiningTime = _startMiningTime;
}
```

图 23 相关函数源码（二）

- **相关函数：**getUnclaimedHLTC、getUnclaimedHDOGE、getUnclaimedHPT、transferOwnership，setEmergencyStop、setStartMiningTime
- **安全建议：无**
- **审计结果：通过**

1. 结论

Beosin(成都链安)对 wata 项目的智能合约的设计和代码实现进行了详细的审计。审计团队在审计过程中发现的问题均已告知项目方并就修复结果达成一致，wata 项目的智能合约的总体审计结果是**通过。**

成都链安

BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号