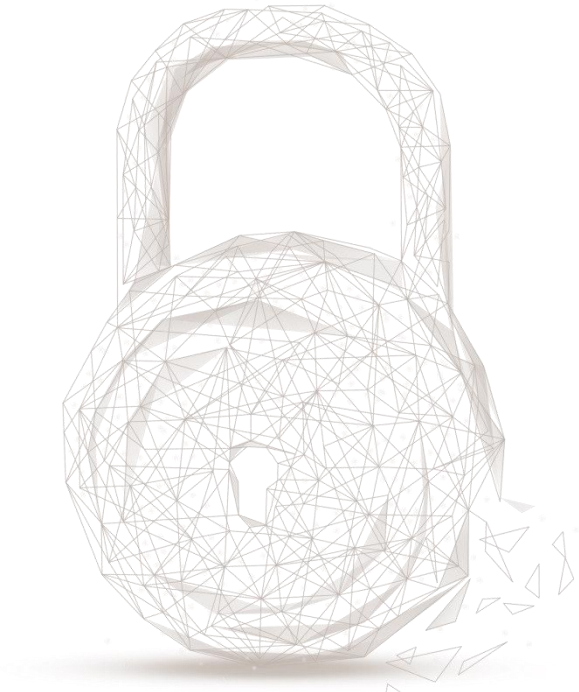
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202108040939

审计合约名称：

C(C)

审计合约地址：

0xFeD613CdD4182b73A032fe1f03Bd7872465010F8

审计合约链接地址：

https://hecoinfo.com/address/0xFeD613CdD4182b73A032fe1f03Bd7872465010F8#code

合约审计开始日期：2021.07.31

合约审计完成日期：2021.08.04

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
		SafeMath 函数审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
		可见性规范审计	通过
		fallback 函数使用审计	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

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C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，C合约通过所有检测，合约审计总体结果为通过。**以下为本合约基本信息。

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. SafeMath函数审计

检查合约中是否正确使用SafeMath库内的函数进行数学运算，或者进行其他防溢出的检查。

- **安全建议：无**
- **审计结果：通过**

5. 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**

6. 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**

7.可见性规范审计

检查合约函数的可见性是否符合设计要求。

- **安全建议：无**
- **审计结果：通过**

8.fallback函数使用审计

检查在当前合约中是否正确使用了fallback函数。

- **安全建议：无**
- **审计结果：通过**

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Heco虚拟机智能合约漏洞，曾导致了TheDAO被攻击。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虚拟机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虚拟机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函数，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

data是由本合约内部构建(不可由用户任意指定)，同时检查了对应的调用返回值，无安全风险。

- **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HRC-20Token的transfer/transferFrom函数实现中，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虚拟机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虚拟机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**

业务审计

1. C代币合约业务函数分析

(1) 代币基本信息

代币名称	C
代币简称	C
代币精度	18
代币总量	1千万亿（可销毁）
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：**C合约实现了符合HRC-20标准的代币。需要说明的是，用户可以直接调用授权函数为指定地址设置授权值，但是为了避免多次授权，建议用户使用increaseAllowance与decreaseAllowance修改授权值。
- **相关函数：** name、symbol、decimals、totalSupply、balanceOf、allowance、transfer、transferFrom、approve、increaseAllowance、decreaseAllowance
- **审计结果：通过**

(3) 记账机制

- **业务描述：**C代币实现了一种与普通代币不同的内部记账机制——t账本与r账本。t账本记录的是用户实际的代币数量，r账本记录的是用户的权重。在查询余额时，普通地址的余额为rOwned除以rate，转换为tOwned。普通地址在转账时，会将转账数量tAmount乘以rate转换为rAmount，然后从账户的rOwned余额中扣除对应数量的rAmount。在代币销毁时，通过将rTotal减去rAmount以

及tTotal减去tAmount实现销毁等比例的代币。在代币分红时，rTotal会减去分红数量的代币rAmount，但是tTotal保持不变，由于rate的值为rTotal除以tTotal，rate会减小，在用户rOwned转换为tOwned时，tOwned的值增大，从而实现用户自动分红。

- **相关函数：**balanceOf, tokenFromReflection, _getRate, _getCurrentSupply, _reflectFee
- **安全建议：无**
- **审计结果：通过**

(4) 分红机制

- **业务描述：**C代币实现了分红机制。每一笔普通交易都会将一定比例(_taxFee，目前为6%，可由owner修改)的代币作为分红。如果交易双方中有手续费例外地址，则该交易不参与分红。
- **相关函数：**transfer, transferFrom, _reflectFee, removeAllFee, restoreAllFee
- **安全建议：无**
- **审计结果：通过**

(5) 排除奖励分红函数

- **业务描述：**合约实现了excludeFromReward函数用于owner将指定地址设置为奖励例外地址（无法获得奖励分红），并更新其当前代币余额；includeInReward函数用于owner将奖励例外地址设置为普通地址（可以获取奖励分红）。奖励例外地址无法使用代币分红，但在转账过程中会将分红记入到账户rOwned余额中。在取消例外地址设置后，因为分红代币已经记入rOwned余额中，可直接使用rOwned对应的余额。注意：在将指定地址设置为例外地址后，过一段时间再修改回来，这段时间的分红认可领取。

```
951     function excludeFromReward(address account) public onlyOwner() {
952         // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude Uniswap router.');
953         require(!_isExcluded[account], "Account is already excluded");
954         if(_rOwned[account] > 0) {
955             _tOwned[account] = tokenFromReflection(_rOwned[account]);
956         }
957         _isExcluded[account] = true;
958         _excluded.push(account);
959     }
960 
961     function includeInReward(address account) external onlyOwner() {
962         require(_isExcluded[account], "Account is already excluded");
963         for (uint256 i = 0; i < _excluded.length; i++) {
964             if (_excluded[i] == account) {
965                 _excluded[i] = _excluded[_excluded.length - 1];
966                 _tOwned[account] = 0;
967                 _isExcluded[account] = false;
968                 _excluded.pop();
969                 break;
970             }
971         }
972     }
```

图 1 excludeFromReward和includeInReward函数源码

- **相关函数：**excludeFromReward，tokenFromReflection，includeInReward
- **安全建议：**require报错信息错误，两个函数条件相反，报错信息相同，建议修改。

- **修复结果：**忽略，不影响合约运行
- **审计结果：通过**

(6) deliver 函数

- **业务描述：**C代币实现了deliver函数，通过该函数，用户可将指定数量的代币从账户中移除，并用于分红。奖励例外地址无法调用该函数。

```
925     function deliver(uint256 tAmount) public {
926         address sender = _msgSender();
927         require(!_isExcluded[sender], "Excluded addresses cannot call this function");
928         (uint256 rAmount,,,,,) = _getValues(tAmount);
929         _rOwned[sender] = _rOwned[sender].sub(rAmount);
930         _rTotal = _rTotal.sub(rAmount);
931         _tFeeTotal = _tFeeTotal.add(tAmount);
932     }
```

图 2 deliver函数源码

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

(7) owner 设置函数

- **业务描述：**C代币实现了excludeFromFee，includeInFee函数用于设置指定地址为手续费例外地址（交易时不收取任何手续费）；setTaxFeePercent，setLiquidityFeePercent函数用于设置交易时的相关手续费比例；setMaxTxPercent函数用owner设置单次交易金额的限制，owner相关交易不受限制；setSwapAndLiquifyEnabled函数用于设置是否在交易时添加流动性。以上函数仅合约的owner可调用。

```
984         function excludeFromFee(address account) public onlyOwner {
985         _isExcludedFromFee[account] = true;
986     }
987
988     function includeInFee(address account) public onlyOwner {
989         _isExcludedFromFee[account] = false;
990     }
991
992     function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
993         _taxFee = taxFee;
994     }
995
996     function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
997         _liquidityFee = liquidityFee;
998     }
999
1000    function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
1001        _maxTxAmount = _tTotal.mul(maxTxPercent).div(
1002            10**2
1003        );
1004    }
1005
1006    function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
1007        swapAndLiquifyEnabled = _enabled;
1008        emit SwapAndLiquifyEnabledUpdated(_enabled);
1009    }
```

图 3 owner设置函数

- **相关函数：**excludeFromFee，includeInFee，setTaxFeePercent，setLiquidityFeePercent，setMaxTxPercent，setSwapAndLiquifyEnabled
- **安全建议：无**
- **审计结果：通过**

(8) 查询函数

- **业务描述：**C合约实现了isExcludedFromReward函数用于查询指定地址是否为奖励例外地址；totalFees函数用于查询最新的总费用；reflectionFromToken和tokenFromReflection用于t账本和r账本的相互转换；isExcludedFromFee函数用于查询指定地址是否为手续费例外地址。

```
917     function isExcludedFromReward(address account) public view returns (bool) {
918         return _isExcluded[account];
919     }
920
921     function totalFees() public view returns (uint256) {
922         return _tFeeTotal;
923     }
```

图 4 相关查询函数（1/3）

```
934     function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256) {
935         require(tAmount <= _tTotal, "Amount must be less than supply");
936         if (!deductTransferFee) {
937             (uint256 rAmount,,,,,) = _getValues(tAmount);
938             return rAmount;
939         } else {
940             (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
941             return rTransferAmount;
942         }
943     }
944 
945     function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
946         require(rAmount <= _rTotal, "Amount must be less than total reflections");
947         uint256 currentRate =  _getRate();
948         return rAmount.div(currentRate);
949     }
```

图 5 相关查询函数（2/3）

```
1092     function isExcludedFromFee(address account) public view returns(bool) {
1093         return _isExcludedFromFee[account];
1094     }
```

图 5 相关查询函数（3/3）

- **相关函数：**isExcludedFromReward，totalFees，reflectionFromToken，tokenFromReflection，isExcludedFromFee
- **安全建议：无**
- **审计结果：通过**

(9) 添加流动性机制

- **业务描述：**C代币实现了添加流动性机制。每一笔普通交易都将一定比例(_liquidityFee，目前为4%，可由owner修改)的代币作为流动性费用由合约收取。如果交易任意一方为手续费例外地址，则该交易不收取流动性费用。当一笔交易满足条件时，会将合约里存储的指定数量代币（numTokensSellToAddToLiquidity）平均分为两份，将一份去MDEX的C-HT交易对交换为HT后，再与剩下的一份去C-HT交易对添加流动性（获得的LP代币将发送给合约的owner）。

```
1126         bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
1127         if (
1128             overMinTokenBalance &&
1129             !inSwapAndLiquify &&
1130             from != uniswapV2Pair &&
1131             swapAndLiquifyEnabled
1132         ) {
1133             contractTokenBalance = numTokensSellToAddToLiquidity;
1134             //add liquidity
1135             swapAndLiquify(contractTokenBalance);
1136         }
```

图 4 _transfer函数源码部分

```
1150    function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1151        // split the contract balance into halves
1152        uint256 half = contractTokenBalance.div(2);
1153        uint256 otherHalf = contractTokenBalance.sub(half);
1154
1155        // capture the contract's current ETH balance.
1156        // this is so that we can capture exactly the amount of ETH that the
1157        // swap creates, and not make the liquidity event include any ETH that
1158        // has been manually sent to the contract
1159        uint256 initialBalance = address(this).balance;
1160
1161        // swap tokens for ETH
1162        swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
1163
1164        // how much ETH did we just swap into?
1165        uint256 newBalance = address(this).balance.sub(initialBalance);
1166
1167        // add liquidity to uniswap
1168        addLiquidity(otherHalf, newBalance);
1169
1170        emit SwapAndLiquify(half, newBalance, otherHalf);
1171    }
```

图 5 swapAndLiquify函数源码

- **相关函数：**transfer，transferFrom
- **安全建议：**swapAndLiquify函数中，将输入的C代币平分，一半通过交换成HT后与剩余的一半C去swap中添加流动性。由于对应的pair合约经交换后价格发生变动（C价格变低），在添加流动性时会导致有极小部分的HT剩余在本合约里且无法取出。并且添加流动性获得的LP代币是发送给合约的owner而不是存储到合约，如果私钥丢失，可能会导致相关交易对流动性被owner地址撤出，造成用户所持代币价值下降。
- **修复结果：**项目方承诺将销毁owner权限。
- **审计结果：通过**

结论

成都链安对C项目的设计和代码实现进行了详细的审计，所有审计过程中发现的问题告知项目方后均已修复或妥善处理。C合约审计的总体结果是**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