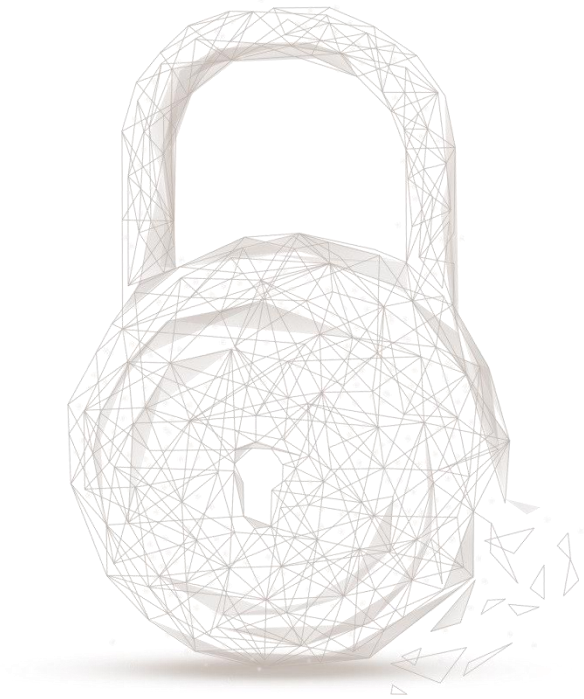
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number: 202108040939

Smart Contract Name:

C (C)

Smart Contract Address:

0xFeD613CdD4182b73A032fe1f03Bd7872465010F8

Smart Contract Address Link:

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

Start Date: 2021.07.31

Completion Date: 2021.08.04

Overall Result: Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass

		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract C, including

Coding Standards, Security, and Business Logic. **The C contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing HT.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level function like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.

- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 Business analysis of Contract C

(1) Basic Token Information

Token name	C
Token symbol	C
decimals	18
totalSupply	1 quadrillion (Burnable)
Token type	HRC-20

Table 1 Basic Token Information

(2) HRC-20 Token Standard Function

- Description: The C contract implements tokens that comply with the HRC-20 standard. It should be noted that the user can directly call the *approve* function to set the approval value for the specified address, but in order to avoid multiple authorizations, it is recommended that the user use *increaseAllowance* and *decreaseAllowance* to modify the authorization value.
- Related function: *name, symbol, decimals, totalSupply, balanceOf, allowance, transfer, transferFrom, approve, increaseAllowance, decreaseAllowance*
- Result: Pass

(3) Bookkeeping mechanism

- Description: C tokens implement an internal bookkeeping mechanism that differs from normal tokens - the t-book and the r-book. t-book records the actual number of tokens a user has, and r-book records the user's weight. When the balance is checked, the balance of the normal address is rOwned divided by rate, which is converted to tOwned. When the normal address is transferred, the number of transfers, tAmount, is converted to rAmount by multiplying it by rate, and then the corresponding amount of rAmount is deducted from the rOwned balance of the account. When the tokens are destroyed, the tokens are converted to rAmount by subtracting rTotal from rAmount and tTotal minus tAmount to destroy an equal proportion of the tokens. When a token is distributed, rTotal is subtracted from the number of tokens distributed, rAmount, but tTotal remains the same. Since the value of rate is rTotal divided by tTotal, rate

decreases and the value of tOwned increases when the user's rOwned is converted to tOwned, thus enabling the user to automatically distribute dividends.

- Related function: *balanceOf*, *tokenFromReflection*, *_getRate*, *_getCurrentSupply*, *_reflectFee*
- Result: Pass

(4) Bonus mechanism

- Description: The C token implements a dividend mechanism. A certain percentage (_taxFee, currently 6%, can be modified by the owner) of the token is taken as a dividend for each normal transaction. If either side of the transaction is a fee exception address, no dividend is paid on that transaction.
- Related function: *transfer*, *transferFrom*, *_reflectFee*, *removeAllFee*, *restoreAllFee*
- Result: Pass

(5) Exclusion of incentive bonus functions

- Description: The contract implements the *excludeFromReward* function which is used by the owner to set the specified address as a reward exception address (no reward dividends are available) and to update its current token balance; the *includeInReward* function is used by the owner to set the reward exception address as a normal address (reward dividends are available). Reward exception addresses cannot use token dividends, but dividends are credited to the account rOwned balance during the transfer process. If owner cancel the exception address setting, user can use the rOwned balance directly, as the bonus tokens are already credited to the rOwned balance. Note: After setting the specified address as an exception address and modifying it back after a period of time, the dividends are approved for collection during this period.

```
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     }
```

Figure 1 source code of *excludeFromReward*, *tokenFromReflection*

- Related function: *excludeFromReward*, *tokenFromReflection*, *includeInReward*
- Safety recommendation: Require error message error, two functions with opposite conditions, same error message, it is recommended to change.
- Repair result: Ignored, does not affect contract operation.

- Result: Pass

(6) deliver function

- Description: The C token implements the *deliver* function, by which the user can remove a specified amount of tAmount from the account and use it for dividends. The function cannot be called from a reward exception address.

```
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     }
```

Figure 2 source code of *deliver*

- Related function: *deliver*
- Result: Pass

(7) Owner setting functions

- Description: C tokens implement the *excludeFromFee*, *includeInFee* functions to set the specified address as the fee exception address (no fee is charged when trading); *setTaxFeePercent*, *setLiquidityFeePercent* functions to set the percentage of fee associated with the transaction. *setMaxTxPercent* function sets the limit of the amount of a single transaction with the owner, the owner's related transactions are not limited; *setSwapAndLiquifyEnabled* function is used to set *setSwapAndLiquifyEnabled* function is used to set whether to add liquidity when trading. The above functions can only be called by the owner of the contract.

```
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    }
```

Figure 3 source code of owner setting functions

- Related function: *excludeFromFee*, *includeInFee*, *setTaxFeePercent*, *setLiquidityFeePercent*, *setMaxTxPercent*, *setSwapAndLiquifyEnabled*
- Result: Pass

(8) Query functions

- Description: C contract implements the *isExcludedFromReward* function for querying whether the specified address is a reward exception address; the *totalFees* function for querying the latest total fees; and the *reflectionFromToken* and *tokenFromReflection* are used to convert t-books and r-books to each other. *isExcludedFromFee* function is used to query whether the specified address is a fee exception address.

```
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     }
```

Figure 4 source code of related query functions(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     }
```

Figure 5 source code of related query functions(2/3)

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

Figure 6 source code of related query functions(3/3)

- Related function: *isExcludedFromReward*, *totalFees*, *reflectionFromToken*, *tokenFromReflection*, *isExcludedFromFee*
- Result: Pass

(9) Adding liquidity mechanism

- Description: The C token implements an added liquidity mechanism. A percentage (_liquidityFee, currently 4%, can be modified by the owner) of the token is charged as a liquidity fee to the contract for each normal transaction. If either party to a transaction is a fee exception address, no liquidity fee is charged for that transaction. When a transaction meets the conditions, the specified number of tokens stored in the contract (numTokensSellToAddToLiquidity) will be divided into two halves, half of the C will be exchanged for HT through the C-HT pair, and then the HT and the remaining half of the C will be added to the C-HT pair as liquidity. (the LP gets the token and sends it to the owner of the contract).

```
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        }
```

Figure 7 source code of _transfer

```
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    }
```

Figure 8 source code of *swapAndLiquify*

- Safety recommendation: In the swapAndLiquify function, the input C tokens are divided equally and half of them are added to the swap by swapping them into HT and then adding liquidity to the remaining half of the C. As the price of the corresponding pair contract changes after the swap (the C price becomes lower), a very small amount of HT remains in the contract and cannot be removed when liquidity is added. Moreover, the LP tokens obtained by adding liquidity are sent to the owner of the contract instead of being stored in the contract. If the private key is lost, this may lead to a decrease in liquidity for the relevant transaction, causing the user to fail to transfer funds.
- Repair result: The project party promised to destroy the owner authority.
- Related function: *transfer*, *transferFrom*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contract C. The C contract passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com