CyberCrime Shield
cybercrimeshield.org

Smart Contract Audit Report

LIFE TOKEN

thelifetoken.com

AUDIT TYPE: **PUBLIC**

https://cybercrimeshield.org/secure/lifetoken
ID:2390328

May 7, 2021

TABLE OF CONTENTS

SMART CONTRACT

https://bscscan.com/address/0x36f66d61db3497f7fdba22efd2a251753a95d0e2#code

Mirror: https://cybercrimeshield.org/secure/uploads/LifeToken.sol

CRC32: 282F54A3

MD5: 65911221D7B642474326A8770F82D285

SHA-1: 23F6BFB8D602ECC47C1D656E1FD35B330799B9C3

INTRODUCTION

Blockchain platforms, such as Nakamoto's Bitcoin, enable the trade of crypto-currencies between mutually mistrusting parties.

To eliminate the need for trust, Nakomoto designed a peer-to-peer network that enables its peers to agree on the trading transactions.

Smart contracts have shown to be applicable in many domains including financial industry, public sector and cross-industry.

The increased adoption of smart contracts demands strong security guarantees. Unfortunately, it is challenging to create smart contracts that are free of security bugs.

As a consequence, critical vulnerabilities in smart contracts are discovered and exploited every few months.

In turn, these exploits have led to losses reaching billions worth of USD in the past few years.

It is apparent that effective security checks for smart contracts are strictly needed.

Our company provides comprehensive, independent smart contract auditing.

We help stakeholders confirm the quality and security of their smart contracts using our standardized audit process.

Life Token is a charity orientated token on the Binance Smart Chain.

The scope of this audit was to analyze and document the Life Token contract.

This document is not financial advice, you perform all financial actions on your own responsibility.

AUDIT METHODOLOGY

1. Design Patterns

We inspect the structure of the smart contract, including both manual and automated analysis.

2. Static Analysis

The static analysis is performed using a series of automated tools, purposefully designed to test the security of the contract.

All the issues found by tools were manually checked (rejected or confirmed).

3. Manual Analysis

Contract reviewing to identify common vulnerabilities. Comparing of requirements and implementation. Reviewing of a smart contract for compliance with specified customer requirements. Checking for energy optimization and self-documentation. Running tests of the properties of the smart contract in test net.

ISSUES DISCOVERED

Issues are listed from most critical to least critical. Severity is determined by an assessment of the risk of exploitation or otherwise unsafe behavior.

Severity Levels

Critical - Funds may be allocated incorrectly, lost or otherwise result in a significant loss.
Medium - Affects the ability of the contract to operate.
Low - Minimal impact on operational ability.
Informational - No impact on the contract.

AUDIT SUMMARY

The summary result of the audit performed is presented in the table below

Findings list:

LEVEL	AMOUNT
Critical	0
Medium	0
Low	4
Informational	5

FINDINGS

SOLIDITY_ADDRESS_HARDCODED
The contract contains unknown address. This address might be used for some malicious activity. Please check hardcoded address and it's usage.
line: 431
column: 8
content: _owner=address(0)

SOLIDITY_ERC20_APPROVE
The <code>approve</code> function of ERC-20 is vulnerable. Using front-running attack one can spend approved tokens before change of <code>allowance</code> value.
line: 533
column: 4
content:
functionapprove(addressspender,uint256amount)publicoverridereturns(bool){_approve(_msgSender(),spender,amount);returntrue;}

SOLIDITY_EXTRA_GAS_IN_LOOPS
State variable, <code>.balance</code>, or <code>.length</code> of non-memory array is used in the condition of <code>for</code> or <code>while</code> loop. In this case, every iteration of loop consumes extra gas.
line: 611
column: 8
content:
for(uint256i=0;i<_excluded.length;i++){if(_excluded[i]==account){_excluded[i]=_excluded[_excluded.length-1];_tOwned[account]=0;_isExcluded[account]=false;_excluded.pop();break;}}

SOLIDITY_EXTRA_GAS_IN_LOOPS
line: 815
column: 8
content:
for(uint256i=0;i<_excluded.length;i++){if(_rOwned[_excluded[i]]>rSupply||_tOwned[_excluded[i]]>tSupply)return(_rTotal,_tTotal);rSupply=rSupply.sub(_rOwned[_excluded[i]]);tSupply=tSupply.sub(_tOwned[_excluded[i]]);}

SOLIDITY_UNCHECKED_CALL

Expect calls to external contract to fail. When sending token, check for the return value and handle errors.

line: 701

column: 8

content: _sendToCharity(tCharity,sender)

SOLIDITY_UNCHECKED_CALL

line: 717

column: 8

content: _sendToCharity(tCharity,sender)

SOLIDITY_UNCHECKED_CALL

patternId: f39eed

severity: 3

line: 735

column: 8

content: _sendToCharity(tCharity,sender)

SOLIDITY_UNCHECKED_CALL

patternId: f39eed

severity: 3

line: 752

column: 8

content: _sendToCharity(tCharity,sender)

SOLIDITY_USING_INLINE_ASSEMBLY

nline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features of Solidity.

line: 279

column: 8

content: assembly{codehash:=extcodehash(account)}

CONCLUSION

- Contract has high code readability
- Gas usage is optimal
- Contract is fully BSC completable
- No backdoors or overflows are present in the contract

Safety guaranteed by CyberCrimeShield Security Company

SOURCE CODE

```
/**
 *Submitted for verification at BscScan.com on 2021-04-16
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.2;


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

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
 * @dev Interface of the BEP20 standard as defined in the EIP.
 */
interface IBEP20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: 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
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external
returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
```

```
         * Note that `value` may be zero.
         */
        event Transfer(address indexed from, address indexed to, uint256 value);

        /**
         * @dev Emitted when the allowance of a `spender` for an `owner` is set by
         * a call to {approve}. `value` is the new allowance.
         */
        event Approval(address indexed owner, address indexed spender, uint256 value);
    }

    /**
     * @dev Wrappers over Solidity's arithmetic operations with added overflow
     * checks.
     *
     * Arithmetic operations in Solidity wrap on overflow. This can easily result
     * in bugs, because programmers usually assume that an overflow raises an
     * error, which is the standard behavior in high level programming languages.
     * `SafeMath` restores this intuition by reverting the transaction when an
     * operation overflows.
     *
     * Using this library instead of the unchecked operations eliminates an entire
     * class of bugs, so it's recommended to use it always.
     */
    library SafeMath {
        /**
         * @dev Returns the addition of two unsigned integers, reverting on
         * overflow.
         *
         * Counterpart to Solidity's `+` operator.
         *
         * Requirements:
         *
         * - Addition cannot overflow.
         */
        function add(uint256 a, uint256 b) internal pure returns (uint256) {
            uint256 c = a + b;
            require(c >= a, "SafeMath: addition overflow");

            return c;
        }

        /**
         * @dev Returns the subtraction of two unsigned integers, reverting on
         * overflow (when the result is negative).
         *
         * Counterpart to Solidity's `-` operator.
         *
         * Requirements:
         *
         * - Subtraction cannot overflow.
         */
        function sub(uint256 a, uint256 b) internal pure returns (uint256) {
            return sub(a, b, "SafeMath: subtraction overflow");
        }

        /**
         * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
         * overflow (when the result is negative).
         *
         * Counterpart to Solidity's `-` operator.
         *
         * Requirements:
         *
         * - Subtraction cannot overflow.
         */
        function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
            require(b <= a, errorMessage);
            uint256 c = a - b;

            return c;
        }

        /**
         * @dev Returns the multiplication of two unsigned integers, reverting on
         * overflow.
         *
         * Counterpart to Solidity's `*` operator.
         *
         * Requirements:
         *
         * - Multiplication cannot 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-contracts/pull/522
            if (a == 0) {
                return 0;
            }

            uint256 c = a * b;
            require(c / a == b, "SafeMath: multiplication overflow");

            return c;
        }

        /**
         * @dev Returns the integer division of two unsigned integers. Reverts on
         * division by zero. The result is rounded towards zero.
         *
         * Counterpart to Solidity's `/` operator. Note: this function uses a
         * `revert` opcode (which leaves remaining gas untouched) while Solidity
         * uses an invalid opcode to revert (consuming all remaining gas).
         *
         * Requirements:
         *
         * - The divisor cannot be zero.
         */
        function div(uint256 a, uint256 b) internal pure returns (uint256) {
            return div(a, b, "SafeMath: division by zero");
        }

        /**
         * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
         * division by zero. The result is rounded towards zero.
         *
         * Counterpart to Solidity's `/` operator. Note: this function uses a
         * `revert` opcode (which leaves remaining gas untouched) while Solidity
         * uses an invalid opcode to revert (consuming all remaining gas).
         *
         * Requirements:
         *
         * - The divisor cannot be zero.
         */
        function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
            require(b > 0, errorMessage);
            uint256 c = a / b;
            // assert(a == b * c + a % b); // There is no case in which this doesn't hold

            return c;
        }

        /**
         * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
         * Reverts when dividing by zero.
         *
         * Counterpart to Solidity's `%` operator. This function uses a `revert`
         * opcode (which leaves remaining gas untouched) while Solidity uses an
         * invalid opcode to revert (consuming all remaining gas).
         *
         * Requirements:
         *
         * - The divisor cannot be zero.
         */
        function mod(uint256 a, uint256 b) internal pure returns (uint256) {
            return mod(a, b, "SafeMath: modulo by zero");
        }

        /**
         * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
         * Reverts with custom message when dividing by zero.
         *
         * Counterpart to Solidity's `%` operator. This function uses a `revert`
         * opcode (which leaves remaining gas untouched) while Solidity uses an
         * invalid opcode to revert (consuming all remaining gas).
         *
         * Requirements:
         *
         * - The divisor cannot be zero.
         */
        function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
            require(b != 0, errorMessage);
            return a % b;
        }
    }

    /**
     * @dev Collection of functions related to the address type
     */
    library Address {
        /**
         * @dev Returns true if `account` is a contract.
         *
         * [IMPORTANT]
         * ====
         * It is unsafe to assume that an address for which this function returns
         * false is an externally-owned account (EOA) and not a contract.
         *
         * Among others, `isContract` will return false for the following
         * types of addresses:
         *
         *  - an externally-owned account
         *  - a contract in construction
         *  - an address where a contract will be created
         *  - an address where a contract lived, but was destroyed
         * ====
         */
        function isContract(address account) internal view returns (bool) {
            // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
            // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
            // for accounts without code, i.e. `keccak256('')`
            bytes32 codehash;
            bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
            // solhint-disable-next-line no-inline-assembly
            assembly { codehash := extcodehash(account) }
            return (codehash != accountHash && codehash != 0x0);
        }

        /**
         * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
         * `recipient`, forwarding all available gas and reverting on errors.
         *
         * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
         * of certain opcodes, possibly making contracts go over the 2300 gas limit
         * imposed by `transfer`, making them unable to receive funds via
         * `transfer`. {sendValue} removes this limitation.
         *
         * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
         *
         * IMPORTANT: because control is transferred to `recipient`, care must be
         * taken to not create reentrancy vulnerabilities. Consider using
         * {ReentrancyGuard} or the
         * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
         */
        function sendValue(address payable recipient, uint256 amount) internal {
            require(address(this).balance >= amount, "Address: insufficient balance");

            // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
            (bool success, ) = recipient.call{ value: amount }("");
            require(success, "Address: unable to send value, recipient may have reverted");
        }

        /**
         * @dev Performs a Solidity function call using a low level `call`. A
         * plain`call` is an unsafe replacement for a function call: use this
         * function instead.
         *
         * If `target` reverts with a revert reason, it is bubbled up by this
         * function (like regular Solidity function calls).
         *
         * Returns the raw returned data. To convert to the expected return value,
         * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
         *
         * Requirements:
         *
         * - `target` must be a contract.
         * - calling `target` with `data` must not revert.
         *
         * _Available since v3.1._
         */
        function functionCall(address target, bytes memory data) internal returns (bytes memory) {
          return functionCall(target, data, "Address: low-level call failed");
        }

        /**
         * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
         * `errorMessage` as a fallback revert reason when `target` reverts.
         *
         * _Available since v3.1._
         */
        function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
            return _functionCallWithValue(target, data, 0, errorMessage);
        }

        /**
```

```
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                        let returndata_size := mload(returndata)
                        revert(add(32, returndata), returndata_size)
                    }
                } else {
                    revert(errorMessage);
                }
            }
        }
}

/**
 * @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.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * 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 public _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);


    /**
     * @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.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract CoinToken is Context, IBEP20, Ownable {
    using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _rOwned;
    mapping (address => uint256) private _tOwned;
    mapping (address => mapping (address => uint256)) private _allowances;

    mapping (address => bool) private _isExcluded;
    mapping (address => bool) private _isCharity;
    address[] private _excluded;

    string  private _NAME;
    string  private _SYMBOL;
    uint256   private _DECIMALS;
    address public FeeAddress;

    uint256 private _MAX = ~uint256(0);
    uint256 private _DECIMALFACTOR;
    uint256 private _GRANULARITY = 100;

    uint256 private _tTotal;
    uint256 private _rTotal;

    uint256 private _tFeeTotal;
    uint256 private _tBurnTotal;
    uint256 private _tCharityTotal;

    uint256 public     _TAX_FEE;
    uint256 public    _BURN_FEE;
    uint256 public _CHARITY_FEE;

    // Track original fees to bypass fees for charity account
    uint256 private ORIG_TAX_FEE;
    uint256 private ORIG_BURN_FEE;
    uint256 private ORIG_CHARITY_FEE;

    constructor (string memory _name, string memory _symbol, uint256 _decimals, uint256 _supply, uint256 _txFee,uint256 _burnFee,uint256 _charityFee,address _FeeAddress,address tokenOwner) {
        _NAME = _name;
        _SYMBOL = _symbol;
        _DECIMALS = _decimals;
        _DECIMALFACTOR = 10 ** uint256(_DECIMALS);
        _tTotal =_supply * _DECIMALFACTOR;
        _rTotal = (_MAX - (_MAX % _tTotal));
        _TAX_FEE = _txFee* 100;
        _BURN_FEE = _burnFee * 100;
        _CHARITY_FEE = _charityFee* 100;
        ORIG_TAX_FEE = _TAX_FEE;
        ORIG_BURN_FEE = _BURN_FEE;
        ORIG_CHARITY_FEE = _CHARITY_FEE;
        _isCharity[_FeeAddress] = true;
        FeeAddress = _FeeAddress;
        _owner = tokenOwner;
        _rOwned[tokenOwner] = _rTotal;

        emit Transfer(address(0),tokenOwner, _tTotal);
    }

    function name() public view returns (string memory) {
        return _NAME;
    }

    function symbol() public view returns (string memory) {
        return _SYMBOL;
    }

    function decimals() public view returns (uint256) {
        return _DECIMALS;
    }

    function totalSupply() public view override returns (uint256) {
        return _tTotal;
    }

    function balanceOf(address account) public view override returns (uint256) {
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "TOKEN20: transfer amount exceeds allowance"));
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
```

```
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "TOKEN20: decreased allowance below zero"));
        return true;
    }

    function isExcluded(address account) public view returns (bool) {
        return _isExcluded[account];
    }

    function isCharity(address account) public view returns (bool) {
        return _isCharity[account];
    }

    function totalFees() public view returns (uint256) {
        return _tFeeTotal;
    }

    function totalBurn() public view returns (uint256) {
        return _tBurnTotal;
    }

    function totalCharity() public view returns (uint256) {
        return _tCharityTotal;
    }

    function deliver(uint256 tAmount) public {
        address sender = _msgSender();
        require(!_isExcluded[sender], "Excluded addresses cannot call this function");
        (uint256 rAmount,,,,,,) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rTotal = _rTotal.sub(rAmount);
        _tFeeTotal = _tFeeTotal.add(tAmount);
    }

    function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256) {
        require(tAmount <= _tTotal, "Amount must be less than supply");
        if (!deductTransferFee) {
            (uint256 rAmount,,,,,,) = _getValues(tAmount);
            return rAmount;
        } else {
            (,uint256 rTransferAmount,,,,,) = _getValues(tAmount);
            return rTransferAmount;
        }
    }

    function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
        require(rAmount <= _rTotal, "Amount must be less than total reflections");
        uint256 currentRate =  _getRate();
        return rAmount.div(currentRate);
    }

    function excludeAccount(address account) external onlyOwner() {
        require(!_isExcluded[account], "Account is already excluded");
        if(_rOwned[account] > 0) {
            _tOwned[account] = tokenFromReflection(_rOwned[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    function includeAccount(address account) external onlyOwner() {
        require(_isExcluded[account], "Account is already excluded");
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                _excluded[i] = _excluded[_excluded.length - 1];
                _tOwned[account] = 0;
                _isExcluded[account] = false;
                _excluded.pop();
                break;
            }
        }
    }

    function setAsCharityAccount(address account) external onlyOwner() {
        require(!_isCharity[account], "Account is already charity account");
        _isCharity[account] = true;
        FeeAddress = account;
    }

    function burn(uint256 _value) public{
        _burn(msg.sender, _value);
    }

    function updateFee(uint256 _txFee,uint256 _burnFee,uint256 _charityFee) onlyOwner() public{
        _TAX_FEE = _txFee* 100;
        _BURN_FEE = _burnFee * 100;
        _CHARITY_FEE = _charityFee* 100;
```

```
            ORIG_TAX_FEE = _TAX_FEE;
            ORIG_BURN_FEE = _BURN_FEE;
            ORIG_CHARITY_FEE = _CHARITY_FEE;
        }


        function _burn(address _who, uint256 _value) internal {
            require(_value <= _rOwned[_who]);
            _rOwned[_who] = _rOwned[_who].sub(_value);
            _tTotal = _tTotal.sub(_value);
            emit Transfer(_who, address(0), _value);
        }

        function mint(address account, uint256 amount) onlyOwner() public {

            _tTotal = _tTotal.add(amount);
            _rOwned[account] = _rOwned[account].add(amount);
            emit Transfer(address(0), account, amount);
        }



        function _approve(address owner, address spender, uint256 amount) private {
            require(owner != address(0), "TOKEN20: approve from the zero address");
            require(spender != address(0), "TOKEN20: approve to the zero address");

            _allowances[owner][spender] = amount;
            emit Approval(owner, spender, amount);
        }

        function _transfer(address sender, address recipient, uint256 amount) private {
            require(sender != address(0), "TOKEN20: transfer from the zero address");
            require(recipient != address(0), "TOKEN20: transfer to the zero address");
            require(amount > 0, "Transfer amount must be greater than zero");

            // Remove fees for transfers to and from charity account or to excluded account
            bool takeFee = true;
            if (_isCharity[sender] || _isCharity[recipient] || _isExcluded[recipient]) {
                takeFee = false;
            }

            if (!takeFee) removeAllFee();


            if (_isExcluded[sender] && !_isExcluded[recipient]) {
                _transferFromExcluded(sender, recipient, amount);
            } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
                _transferToExcluded(sender, recipient, amount);
            } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
                _transferStandard(sender, recipient, amount);
            } else if (_isExcluded[sender] && _isExcluded[recipient]) {
                _transferBothExcluded(sender, recipient, amount);
            } else {
                _transferStandard(sender, recipient, amount);
            }

            if (!takeFee) restoreAllFee();
        }

        function _transferStandard(address sender, address recipient, uint256 tAmount) private
    {
            uint256 currentRate =  _getRate();
            (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
    uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
            uint256 rBurn =  tBurn.mul(currentRate);
            uint256 rCharity = tCharity.mul(currentRate);
            _standardTransferContent(sender, recipient, rAmount, rTransferAmount);
            _sendToCharity(tCharity, sender);
            _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
            emit Transfer(sender, recipient, tTransferAmount);
        }

        function _standardTransferContent(address sender, address recipient, uint256 rAmount,
    uint256 rTransferAmount) private {
            _rOwned[sender] = _rOwned[sender].sub(rAmount);
            _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        }

        function _transferToExcluded(address sender, address recipient, uint256 tAmount)
    private {
            uint256 currentRate =  _getRate();
            (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
    uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
            uint256 rBurn =  tBurn.mul(currentRate);
            uint256 rCharity = tCharity.mul(currentRate);
            _excludedFromTransferContent(sender, recipient, tTransferAmount, rAmount,
    rTransferAmount);
            _sendToCharity(tCharity, sender);
            _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
            emit Transfer(sender, recipient, tTransferAmount);
    }

    function _excludedFromTransferContent(address sender, address recipient, uint256 tTransferAmount, uint256 rAmount, uint256 rTransferAmount) private {
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    }


    function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
        uint256 rBurn =  tBurn.mul(currentRate);
        uint256 rCharity = tCharity.mul(currentRate);
        _excludedToTransferContent(sender, recipient, tAmount, rAmount, rTransferAmount);
        _sendToCharity(tCharity, sender);
        _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _excludedToTransferContent(address sender, address recipient, uint256 tAmount, uint256 rAmount, uint256 rTransferAmount) private {
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    }

    function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
        uint256 rBurn =  tBurn.mul(currentRate);
        uint256 rCharity = tCharity.mul(currentRate);
        _bothTransferContent(sender, recipient, tAmount, rAmount, tTransferAmount, rTransferAmount);
        _sendToCharity(tCharity, sender);
        _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _bothTransferContent(address sender, address recipient, uint256 tAmount, uint256 rAmount, uint256 tTransferAmount, uint256 rTransferAmount) private {
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    }

    function _reflectFee(uint256 rFee, uint256 rBurn, uint256 rCharity, uint256 tFee, uint256 tBurn, uint256 tCharity) private {
        _rTotal = _rTotal.sub(rFee).sub(rBurn).sub(rCharity);
        _tFeeTotal = _tFeeTotal.add(tFee);
        _tBurnTotal = _tBurnTotal.add(tBurn);
        _tCharityTotal = _tCharityTotal.add(tCharity);
        _tTotal = _tTotal.sub(tBurn);
        emit Transfer(address(this), address(0), tBurn);
    }


    function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256, uint256, uint256) {
        (uint256 tFee, uint256 tBurn, uint256 tCharity) = _getTBasics(tAmount, _TAX_FEE, _BURN_FEE, _CHARITY_FEE);
        uint256 tTransferAmount = getTTransferAmount(tAmount, tFee, tBurn, tCharity);
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rFee) = _getRBasics(tAmount, tFee, currentRate);
        uint256 rTransferAmount = _getRTransferAmount(rAmount, rFee, tBurn, tCharity, currentRate);
        return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tBurn, tCharity);
    }

    function _getTBasics(uint256 tAmount, uint256 taxFee, uint256 burnFee, uint256 charityFee) private view returns (uint256, uint256, uint256) {
        uint256 tFee = ((tAmount.mul(taxFee)).div(_GRANULARITY)).div(100);
        uint256 tBurn = ((tAmount.mul(burnFee)).div(_GRANULARITY)).div(100);
        uint256 tCharity = ((tAmount.mul(charityFee)).div(_GRANULARITY)).div(100);
        return (tFee, tBurn, tCharity);
    }

    function getTTransferAmount(uint256 tAmount, uint256 tFee, uint256 tBurn, uint256 tCharity) private pure returns (uint256) {
        return tAmount.sub(tFee).sub(tBurn).sub(tCharity);
    }

    function _getRBasics(uint256 tAmount, uint256 tFee, uint256 currentRate) private pure returns (uint256, uint256) {
        uint256 rAmount = tAmount.mul(currentRate);
        uint256 rFee = tFee.mul(currentRate);
        return (rAmount, rFee);
    }

    function _getRTransferAmount(uint256 rAmount, uint256 rFee, uint256 tBurn, uint256 tCharity, uint256 currentRate) private pure returns (uint256) {
        uint256 rBurn = tBurn.mul(currentRate);
        uint256 rCharity = tCharity.mul(currentRate);
        uint256 rTransferAmount = rAmount.sub(rFee).sub(rBurn).sub(rCharity);
        return rTransferAmount;
    }

    function _getRate() private view returns(uint256) {
        (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
        return rSupply.div(tSupply);
    }

    function _getCurrentSupply() private view returns(uint256, uint256) {
        uint256 rSupply = _rTotal;
        uint256 tSupply = _tTotal;
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
            rSupply = rSupply.sub(_rOwned[_excluded[i]]);
            tSupply = tSupply.sub(_tOwned[_excluded[i]]);
        }
        if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
        return (rSupply, tSupply);
    }

    function _sendToCharity(uint256 tCharity, address sender) private {
        uint256 currentRate = _getRate();
        uint256 rCharity = tCharity.mul(currentRate);
        _rOwned[FeeAddress] = _rOwned[FeeAddress].add(rCharity);
        _tOwned[FeeAddress] = _tOwned[FeeAddress].add(tCharity);
        emit Transfer(sender, FeeAddress, tCharity);
    }

    function removeAllFee() private {
        if(_TAX_FEE == 0 && _BURN_FEE == 0 && _CHARITY_FEE == 0) return;

        ORIG_TAX_FEE = _TAX_FEE;
```

```
            ORIG_BURN_FEE = _BURN_FEE;
            ORIG_CHARITY_FEE = _CHARITY_FEE;

            _TAX_FEE = 0;
            _BURN_FEE = 0;
            _CHARITY_FEE = 0;
        }

        function restoreAllFee() private {
            _TAX_FEE = ORIG_TAX_FEE;
            _BURN_FEE = ORIG_BURN_FEE;
            _CHARITY_FEE = ORIG_CHARITY_FEE;
        }

        function _getTaxFee() private view returns(uint256) {
            return _TAX_FEE;
        }


    }
```