



CyberCrime Shield

cybercrimeshield.org

# Smart Contract Audit Report

# LIFE TOKEN

[thelifetoken.com](https://thelifetoken.com)

AUDIT TYPE: **PUBLIC**



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

May 7, 2021



## TABLE OF CONTENTS

SMART CONTRACT.....	3
INTRODUCTION.....	4
AUDIT METHODOLOGY.....	5
ISSUES DISCOVERED.....	6
AUDIT SUMMARY.....	6
FINDINGS.....	7
CONCLUSION.....	8
SOURCE CODE.....	9



# CyberCrime Shield

cybercrimeshield.org

## **SMART CONTRACT**

<https://bscscan.com/address/0x36f66d61db3497f7fdb22efd2a251753a95d0e2#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 `approve` function of ERC-20 is vulnerable. Using front-running attack one can spend approved tokens before change of `allowance` value.

line: 533

column: 4

content:

```
functionapprove(addressspender,uint256amount)publicoverridereturns(bool){_approve(_msgSender(),spender,amount);returntrue;}
```

### SOLIDITY\_EXTRA\_GAS\_IN\_LOOPS

State variable, `.balance`, or `.length` of non-memory array is used in the condition of `for` or `while` 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]]);}
```



# CyberCrime Shield

cybercrimeshield.org

## 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

inline 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





## SOURCE CODE

```
1.  /**
2.   *Submitted for verification at BscScan.com on 2021-04-16
3.   */
4.
5.   // SPDX-License-Identifier: MIT
6.
7.   pragma solidity ^0.8.2;
8.
9.
10.  abstract contract Context {
11.      function _msgSender() internal view virtual returns (address payable) {
12.          return payable(msg.sender);
13.      }
14.
15.      function _msgData() internal view virtual returns (bytes memory) {
16.          this; // silence state mutability warning without generating bytecode -
17.          see https://github.com/ethereum/solidity/issues/2691
18.          return msg.data;
19.      }
20.  }
21.  /**
22.   * @dev Interface of the BEP20 standard as defined in the EIP.
23.   */
24.  interface IBEP20 {
25.      /**
26.       * @dev Returns the amount of tokens in existence.
27.       */
28.      function totalSupply() external view returns (uint256);
29.
30.      /**
31.       * @dev Returns the amount of tokens owned by `account`.
32.       */
33.      function balanceOf(address account) external view returns (uint256);
34.
35.      /**
36.       * @dev Moves `amount` tokens from the caller's account to `recipient`.
37.       *
38.       * Returns a boolean value indicating whether the operation succeeded.
39.       */
```



# CyberCrime Shield

cybercrimeshield.org

```
40.     * Emits a {Transfer} event.
41.     */
42.     function transfer(address recipient, uint256 amount) external returns (bool);
43.
44.     /**
45.     * @dev Returns the remaining number of tokens that `spender` will be
46.     * allowed to spend on behalf of `owner` through {transferFrom}. This is
47.     * zero by default.
48.     *
49.     * This value changes when {approve} or {transferFrom} are called.
50.     */
51.     function allowance(address owner, address spender) external view returns (uint256);
52.
53.     /**
54.     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
55.     *
56.     * Returns a boolean value indicating whether the operation succeeded.
57.     *
58.     * IMPORTANT: Beware that changing an allowance with this method brings the risk
59.     * that someone may use both the old and the new allowance by unfortunate
60.     * transaction ordering. One possible solution to mitigate this race
61.     * condition is to first reduce the spender's allowance to 0 and set the
62.     * desired value afterwards:
63.     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
64.     *
65.     * Emits an {Approval} event.
66.     */
67.     function approve(address spender, uint256 amount) external returns (bool);
68.
69.     /**
70.     * @dev Moves `amount` tokens from `sender` to `recipient` using the
71.     * allowance mechanism. `amount` is then deducted from the caller's
72.     * allowance.
73.     *
74.     * Returns a boolean value indicating whether the operation succeeded.
75.     *
76.     * Emits a {Transfer} event.
77.     */
78.     function transferFrom(address sender, address recipient, uint256 amount) external
79.     returns (bool);
80.
81.     /**
82.     * @dev Emitted when `value` tokens are moved from one account (`from`) to
83.     * another (`to`).
```



# CyberCrime Shield

cybercrimeshield.org

```
84.     * Note that `value` may be zero.
85.     */
86.     event Transfer(address indexed from, address indexed to, uint256 value);
87.
88.     /**
89.     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
90.     * a call to {approve}. `value` is the new allowance.
91.     */
92.     event Approval(address indexed owner, address indexed spender, uint256 value);
93. }
94.
95. /**
96. * @dev Wrappers over Solidity's arithmetic operations with added overflow
97. * checks.
98. *
99. * Arithmetic operations in Solidity wrap on overflow. This can easily result
100. * in bugs, because programmers usually assume that an overflow raises an
101. * error, which is the standard behavior in high level programming languages.
102. * `SafeMath` restores this intuition by reverting the transaction when an
103. * operation overflows.
104. *
105. * Using this library instead of the unchecked operations eliminates an entire
106. * class of bugs, so it's recommended to use it always.
107. */
108. library SafeMath {
109.     /**
110.     * @dev Returns the addition of two unsigned integers, reverting on
111.     * overflow.
112.     *
113.     * Counterpart to Solidity's `+` operator.
114.     *
115.     * Requirements:
116.     *
117.     * - Addition cannot overflow.
118.     */
119.     function add(uint256 a, uint256 b) internal pure returns (uint256) {
120.         uint256 c = a + b;
121.         require(c >= a, "SafeMath: addition overflow");
122.
123.         return c;
124.     }
125.
126.     /**
127.     * @dev Returns the subtraction of two unsigned integers, reverting on
128.     * overflow (when the result is negative).
```



# CyberCrime Shield

cybercrimeshield.org

```
129.     *
130.     * Counterpart to Solidity's `~` operator.
131.     *
132.     * Requirements:
133.     *
134.     * - Subtraction cannot overflow.
135.     */
136.     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
137.         return sub(a, b, "SafeMath: subtraction overflow");
138.     }
139.
140.     /**
141.     * @dev Returns the subtraction of two unsigned integers, reverting with custom
    message on
142.     * overflow (when the result is negative).
143.     *
144.     * Counterpart to Solidity's `~` operator.
145.     *
146.     * Requirements:
147.     *
148.     * - Subtraction cannot overflow.
149.     */
150.     function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns
    (uint256) {
151.         require(b <= a, errorMessage);
152.         uint256 c = a - b;
153.
154.         return c;
155.     }
156.
157.     /**
158.     * @dev Returns the multiplication of two unsigned integers, reverting on
159.     * overflow.
160.     *
161.     * Counterpart to Solidity's `*` operator.
162.     *
163.     * Requirements:
164.     *
165.     * - Multiplication cannot overflow.
166.     */
167.     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
168.         // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
169.         // benefit is lost if 'b' is also tested.
170.         // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
171.         if (a == 0) {
```



```
172.         return 0;
173.     }
174.
175.     uint256 c = a * b;
176.     require(c / a == b, "SafeMath: multiplication overflow");
177.
178.     return c;
179. }
180.
181. /**
182.  * @dev Returns the integer division of two unsigned integers. Reverts on
183.  * division by zero. The result is rounded towards zero.
184.  *
185.  * Counterpart to Solidity's `/` operator. Note: this function uses a
186.  * `revert` opcode (which leaves remaining gas untouched) while Solidity
187.  * uses an invalid opcode to revert (consuming all remaining gas).
188.  *
189.  * Requirements:
190.  *
191.  * - The divisor cannot be zero.
192.  */
193. function div(uint256 a, uint256 b) internal pure returns (uint256) {
194.     return div(a, b, "SafeMath: division by zero");
195. }
196.
197. /**
198.  * @dev Returns the integer division of two unsigned integers. Reverts with custom
199.  * message on
200.  *
201.  * division by zero. The result is rounded towards zero.
202.  *
203.  * Counterpart to Solidity's `/` operator. Note: this function uses a
204.  * `revert` opcode (which leaves remaining gas untouched) while Solidity
205.  * uses an invalid opcode to revert (consuming all remaining gas).
206.  *
207.  * Requirements:
208.  *
209.  * - The divisor cannot be zero.
210.  */
211. function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns
212. (uint256) {
213.     require(b > 0, errorMessage);
214.     uint256 c = a / b;
215.     // assert(a == b * c + a % b); // There is no case in which this doesn't hold
216.
217.     return c;
218. }
```



# CyberCrime Shield

cybercrimeshield.org

```
215.     }
216.
217.     /**
218.      * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
219.      * Reverts when dividing by zero.
220.      *
221.      * Counterpart to Solidity's `%` operator. This function uses a `revert`
222.      * opcode (which leaves remaining gas untouched) while Solidity uses an
223.      * invalid opcode to revert (consuming all remaining gas).
224.      *
225.      * Requirements:
226.      *
227.      * - The divisor cannot be zero.
228.      */
229.     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
230.         return mod(a, b, "SafeMath: modulo by zero");
231.     }
232.
233.     /**
234.      * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
235.      * Reverts with custom message when dividing by zero.
236.      *
237.      * Counterpart to Solidity's `%` operator. This function uses a `revert`
238.      * opcode (which leaves remaining gas untouched) while Solidity uses an
239.      * invalid opcode to revert (consuming all remaining gas).
240.      *
241.      * Requirements:
242.      *
243.      * - The divisor cannot be zero.
244.      */
245.     function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
246.         require(b != 0, errorMessage);
247.         return a % b;
248.     }
249. }
250.
251. /**
252.  * @dev Collection of functions related to the address type
253.  */
254. library Address {
255.     /**
256.      * @dev Returns true if `account` is a contract.
```



# CyberCrime Shield

cybercrimeshield.org

```
257.      *
258.      * [IMPORTANT]
259.      * ====
260.      * It is unsafe to assume that an address for which this function returns
261.      * false is an externally-owned account (EOA) and not a contract.
262.      *
263.      * Among others, `isContract` will return false for the following
264.      * types of addresses:
265.      *
266.      * - an externally-owned account
267.      * - a contract in construction
268.      * - an address where a contract will be created
269.      * - an address where a contract lived, but was destroyed
270.      * ====
271.      */
272.      function isContract(address account) internal view returns (bool) {
273.          // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
274.          // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
275.          // for accounts without code, i.e. `keccak256(')`
276.          bytes32 codehash;
277.          bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
278.          // solhint-disable-next-line no-inline-assembly
279.          assembly { codehash := extcodehash(account) }
280.          return (codehash != accountHash && codehash != 0x0);
281.      }
282.
283.      /**
284.      * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
285.      * `recipient`, forwarding all available gas and reverting on errors.
286.      *
287.      * https://eips.ethereum.org/EIPS/eip-1884 [EIP1884] increases the gas cost
288.      * of certain opcodes, possibly making contracts go over the 2300 gas limit
289.      * imposed by `transfer`, making them unable to receive funds via
290.      * `transfer`. {sendValue} removes this limitation.
291.      *
292.      * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-
now/[Learn more].
293.      *
294.      * IMPORTANT: because control is transferred to `recipient`, care must be
295.      * taken to not create reentrancy vulnerabilities. Consider using
296.      * {ReentrancyGuard} or the
297.      * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-
checks-effects-interactions-pattern[checks-effects-interactions pattern].
```





# CyberCrime Shield

cybercrimeshield.org

```
298.     */
299.     function sendValue(address payable recipient, uint256 amount) internal {
300.         require(address(this).balance >= amount, "Address: insufficient balance");
301.
302.         // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
303.         (bool success, ) = recipient.call{ value: amount }("");
304.         require(success, "Address: unable to send value, recipient may have reverted");
305.     }
306.
307.     /**
308.      * @dev Performs a Solidity function call using a low level `call`. A
309.      * plain `call` is an unsafe replacement for a function call: use this
310.      * function instead.
311.      *
312.      * If `target` reverts with a revert reason, it is bubbled up by this
313.      * function (like regular Solidity function calls).
314.      *
315.      * Returns the raw returned data. To convert to the expected return value,
316.      * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions\[abi.decode`\].
317.      *
318.      * Requirements:
319.      *
320.      * - `target` must be a contract.
321.      * - calling `target` with `data` must not revert.
322.      *
323.      * _Available since v3.1._
324.      */
325.     function functionCall(address target, bytes memory data) internal returns (bytes
memory) {
326.         return functionCall(target, data, "Address: low-level call failed");
327.     }
328.
329.     /**
330.      * @dev Same as {xref-Address-functionCall-address-bytes-}\[`functionCall`\], but with
331.      * `errorMessage` as a fallback revert reason when `target` reverts.
332.      *
333.      * _Available since v3.1._
334.      */
335.     function functionCall(address target, bytes memory data, string memory errorMessage)
internal returns (bytes memory) {
336.         return _functionCallWithValue(target, data, 0, errorMessage);
337.     }
338.
339.     /**
```



# CyberCrime Shield

cybercrimeshield.org

```
340.     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
341.     * but also transferring `value` wei to `target`.
342.     *
343.     * Requirements:
344.     *
345.     * - the calling contract must have an ETH balance of at least `value`.
346.     * - the called Solidity function must be `payable`.
347.     *
348.     * _Available since v3.1._
349.     */
350.     function functionCallWithValue(address target, bytes memory data, uint256 value)
internal returns (bytes memory) {
351.         return functionCallWithValue(target, data, value, "Address: low-level call with
value failed");
352.     }
353.
354.     /**
355.     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-
}[`functionCallWithValue`], but
356.     * with `errorMessage` as a fallback revert reason when `target` reverts.
357.     *
358.     * _Available since v3.1._
359.     */
360.     function functionCallWithValue(address target, bytes memory data, uint256 value,
string memory errorMessage) internal returns (bytes memory) {
361.         require(address(this).balance >= value, "Address: insufficient balance for call");
362.         return _functionCallWithValue(target, data, value, errorMessage);
363.     }
364.
365.     function _functionCallWithValue(address target, bytes memory data, uint256 weiValue,
string memory errorMessage) private returns (bytes memory) {
366.         require(isContract(target), "Address: call to non-contract");
367.
368.         // solhint-disable-next-line avoid-low-level-calls
369.         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
370.         if (success) {
371.             return returndata;
372.         } else {
373.             // Look for revert reason and bubble it up if present
374.             if (returndata.length > 0) {
375.                 // The easiest way to bubble the revert reason is using memory via
assembly
376.
377.                 // solhint-disable-next-line no-inline-assembly
378.                 assembly {
```



# CyberCrime Shield

cybercrimeshield.org

```
379.         let returndata_size := mload(returndata)
380.         revert(add(32, returndata), returndata_size)
381.     }
382.     } else {
383.         revert(errorMessage);
384.     }
385. }
386. }
387. }
388.
389. /**
390.  * @dev Contract module which provides a basic access control mechanism, where
391.  * there is an account (an owner) that can be granted exclusive access to
392.  * specific functions.
393.  *
394.  * By default, the owner account will be the one that deploys the contract. This
395.  * can later be changed with {transferOwnership}.
396.  *
397.  * This module is used through inheritance. It will make available the modifier
398.  * `onlyOwner`, which can be applied to your functions to restrict their use to
399.  * the owner.
400.  */
401. contract Ownable is Context {
402.     address public _owner;
403.
404.     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
405.
406.
407.     /**
408.      * @dev Returns the address of the current owner.
409.      */
410.     function owner() public view returns (address) {
411.         return _owner;
412.     }
413.
414.     /**
415.      * @dev Throws if called by any account other than the owner.
416.      */
417.     modifier onlyOwner() {
418.         require(_owner == _msgSender(), "Ownable: caller is not the owner");
419.         _;
420.     }
421.
422.     /**
423.      * @dev Leaves the contract without owner. It will not be possible to call
```



# CyberCrime Shield

cybercrimeshield.org

```
424.     * `onlyOwner` functions anymore. Can only be called by the current owner.
425.     *
426.     * NOTE: Renouncing ownership will leave the contract without an owner,
427.     * thereby removing any functionality that is only available to the owner.
428.     */
429.     function renounceOwnership() public virtual onlyOwner {
430.         emit OwnershipTransferred(_owner, address(0));
431.         _owner = address(0);
432.     }
433.
434.     /**
435.     * @dev Transfers ownership of the contract to a new account (`newOwner`).
436.     * Can only be called by the current owner.
437.     */
438.     function transferOwnership(address newOwner) public virtual onlyOwner {
439.         require(newOwner != address(0), "Ownable: new owner is the zero address");
440.         emit OwnershipTransferred(_owner, newOwner);
441.         _owner = newOwner;
442.     }
443. }
444.
445. contract CoinToken is Context, IBEP20, Ownable {
446.     using SafeMath for uint256;
447.     using Address for address;
448.
449.     mapping (address => uint256) private _rOwned;
450.     mapping (address => uint256) private _tOwned;
451.     mapping (address => mapping (address => uint256)) private _allowances;
452.
453.     mapping (address => bool) private _isExcluded;
454.     mapping (address => bool) private _isCharity;
455.     address[] private _excluded;
456.
457.     string private _NAME;
458.     string private _SYMBOL;
459.     uint256 private _DECIMALS;
460.     address public FeeAddress;
461.
462.     uint256 private _MAX = ~uint256(0);
463.     uint256 private _DECIMALFACTOR;
464.     uint256 private _GRANULARITY = 100;
465.
466.     uint256 private _tTotal;
467.     uint256 private _rTotal;
468.
```



# CyberCrime Shield

cybercrimeshield.org

```
469.     uint256 private _tFeeTotal;
470.     uint256 private _tBurnTotal;
471.     uint256 private _tCharityTotal;
472.
473.     uint256 public     _TAX_FEE;
474.     uint256 public     _BURN_FEE;
475.     uint256 public     _CHARITY_FEE;
476.
477.     // Track original fees to bypass fees for charity account
478.     uint256 private ORIG_TAX_FEE;
479.     uint256 private ORIG_BURN_FEE;
480.     uint256 private ORIG_CHARITY_FEE;
481.
482.     constructor (string memory _name, string memory _symbol, uint256 _decimals, uint256
        _supply, uint256 _txFee,uint256 _burnFee,uint256 _charityFee,address _FeeAddress,address
        tokenOwner) {
483.         _NAME = _name;
484.         _SYMBOL = _symbol;
485.         _DECIMALS = _decimals;
486.         _DECIMALFACTOR = 10 ** uint256(_DECIMALS);
487.         _tTotal = _supply * _DECIMALFACTOR;
488.         _rTotal = (_MAX - (_MAX % _tTotal));
489.         _TAX_FEE = _txFee* 100;
490.         _BURN_FEE = _burnFee * 100;
491.         _CHARITY_FEE = _charityFee* 100;
492.         ORIG_TAX_FEE = _TAX_FEE;
493.         ORIG_BURN_FEE = _BURN_FEE;
494.         ORIG_CHARITY_FEE = _CHARITY_FEE;
495.         _isCharity[_FeeAddress] = true;
496.         FeeAddress = _FeeAddress;
497.         _owner = tokenOwner;
498.         _rOwned[tokenOwner] = _rTotal;
499.
500.         emit Transfer(address(0),tokenOwner, _tTotal);
501.     }
502.
503.     function name() public view returns (string memory) {
504.         return _NAME;
505.     }
506.
507.     function symbol() public view returns (string memory) {
508.         return _SYMBOL;
509.     }
510.
511.     function decimals() public view returns (uint256) {
```



# CyberCrime Shield

cybercrimeshield.org

```
512.         return _DECIMALS;
513.     }
514.
515.     function totalSupply() public view override returns (uint256) {
516.         return _tTotal;
517.     }
518.
519.     function balanceOf(address account) public view override returns (uint256) {
520.         if (_isExcluded[account]) return _tOwned[account];
521.         return tokenFromReflection(_rOwned[account]);
522.     }
523.
524.     function transfer(address recipient, uint256 amount) public override returns (bool) {
525.         _transfer(_msgSender(), recipient, amount);
526.         return true;
527.     }
528.
529.     function allowance(address owner, address spender) public view override returns
(uint256) {
530.         return _allowances[owner][spender];
531.     }
532.
533.     function approve(address spender, uint256 amount) public override returns (bool) {
534.         _approve(_msgSender(), spender, amount);
535.         return true;
536.     }
537.
538.     function transferFrom(address sender, address recipient, uint256 amount) public
override returns (bool) {
539.         _transfer(sender, recipient, amount);
540.         _approve(sender, _msgSender(),
_allowances[sender][_msgSender()].sub(amount, "TOKEN20: transfer amount exceeds
allowance"));
541.         return true;
542.     }
543.
544.     function increaseAllowance(address spender, uint256 addedValue) public virtual returns
(bool) {
545.         _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].add(addedValue));
546.         return true;
547.     }
548.
549.     function decreaseAllowance(address spender, uint256 subtractedValue) public virtual
returns (bool) {
```



# CyberCrime Shield

cybercrimeshield.org

```
550.     _approve(_msgSender(), spender,
        _allowances[_msgSender()][spender].sub(subtractedValue, "TOKEN20: decreased allowance
        below zero"));
551.         return true;
552.     }
553.
554.     function isExcluded(address account) public view returns (bool) {
555.         return _isExcluded[account];
556.     }
557.
558.     function isCharity(address account) public view returns (bool) {
559.         return _isCharity[account];
560.     }
561.
562.     function totalFees() public view returns (uint256) {
563.         return _tFeeTotal;
564.     }
565.
566.     function totalBurn() public view returns (uint256) {
567.         return _tBurnTotal;
568.     }
569.
570.     function totalCharity() public view returns (uint256) {
571.         return _tCharityTotal;
572.     }
573.
574.     function deliver(uint256 tAmount) public {
575.         address sender = _msgSender();
576.         require(!_isExcluded[sender], "Excluded addresses cannot call this function");
577.         (uint256 rAmount,,,,,) = _getValues(tAmount);
578.         _rOwned[sender] = _rOwned[sender].sub(rAmount);
579.         _rTotal = _rTotal.sub(rAmount);
580.         _tFeeTotal = _tFeeTotal.add(tAmount);
581.     }
582.
583.     function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view
        returns(uint256) {
584.         require(tAmount <= _tTotal, "Amount must be less than supply");
585.         if (!deductTransferFee) {
586.             (uint256 rAmount,,,,,) = _getValues(tAmount);
587.             return rAmount;
588.         } else {
589.             (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
590.             return rTransferAmount;
591.         }
    }
```



# CyberCrime Shield

cybercrimeshield.org

```
592.     }
593.
594.     function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
595.         require(rAmount <= _rTotal, "Amount must be less than total reflections");
596.         uint256 currentRate = _getRate();
597.         return rAmount.div(currentRate);
598.     }
599.
600.     function excludeAccount(address account) external onlyOwner() {
601.         require(!_isExcluded[account], "Account is already excluded");
602.         if (_rOwned[account] > 0) {
603.             _tOwned[account] = tokenFromReflection(_rOwned[account]);
604.         }
605.         _isExcluded[account] = true;
606.         _excluded.push(account);
607.     }
608.
609.     function includeAccount(address account) external onlyOwner() {
610.         require(_isExcluded[account], "Account is already excluded");
611.         for (uint256 i = 0; i < _excluded.length; i++) {
612.             if (_excluded[i] == account) {
613.                 _excluded[i] = _excluded[_excluded.length - 1];
614.                 _tOwned[account] = 0;
615.                 _isExcluded[account] = false;
616.                 _excluded.pop();
617.                 break;
618.             }
619.         }
620.     }
621.
622.     function setAsCharityAccount(address account) external onlyOwner() {
623.         require(!_isCharity[account], "Account is already charity account");
624.         _isCharity[account] = true;
625.         FeeAddress = account;
626.     }
627.
628.     function burn(uint256 _value) public{
629.         _burn(msg.sender, _value);
630.     }
631.
632.     function updateFee(uint256 _txFee,uint256 _burnFee,uint256 _charityFee) onlyOwner()
        public{
633.         _TAX_FEE = _txFee* 100;
634.         _BURN_FEE = _burnFee * 100;
635.         _CHARITY_FEE = _charityFee* 100;
```





# CyberCrime Shield

cybercrimeshield.org

```
636.         ORIG_TAX_FEE = _TAX_FEE;
637.         ORIG_BURN_FEE = _BURN_FEE;
638.         ORIG_CHARITY_FEE = _CHARITY_FEE;
639.     }
640.
641.
642.     function _burn(address _who, uint256 _value) internal {
643.         require(_value <= _rOwned[_who]);
644.         _rOwned[_who] = _rOwned[_who].sub(_value);
645.         _tTotal = _tTotal.sub(_value);
646.         emit Transfer(_who, address(0), _value);
647.     }
648.
649.     function mint(address account, uint256 amount) onlyOwner() public {
650.
651.         _tTotal = _tTotal.add(amount);
652.         _rOwned[account] = _rOwned[account].add(amount);
653.         emit Transfer(address(0), account, amount);
654.     }
655.
656.
657.
658.     function _approve(address owner, address spender, uint256 amount) private {
659.         require(owner != address(0), "TOKEN20: approve from the zero address");
660.         require(spender != address(0), "TOKEN20: approve to the zero address");
661.
662.         _allowances[owner][spender] = amount;
663.         emit Approval(owner, spender, amount);
664.     }
665.
666.     function _transfer(address sender, address recipient, uint256 amount) private {
667.         require(sender != address(0), "TOKEN20: transfer from the zero address");
668.         require(recipient != address(0), "TOKEN20: transfer to the zero address");
669.         require(amount > 0, "Transfer amount must be greater than zero");
670.
671.         // Remove fees for transfers to and from charity account or to excluded account
672.         bool takeFee = true;
673.         if (_isCharity[sender] || _isCharity[recipient] || _isExcluded[recipient]) {
674.             takeFee = false;
675.         }
676.
677.         if (!takeFee) removeAllFee();
678.
679.
680.         if (_isExcluded[sender] && !_isExcluded[recipient]) {
```



# CyberCrime Shield

cybercrimeshield.org

```
681.         _transferFromExcluded(sender, recipient, amount);
682.     } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
683.         _transferToExcluded(sender, recipient, amount);
684.     } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
685.         _transferStandard(sender, recipient, amount);
686.     } else if (_isExcluded[sender] && _isExcluded[recipient]) {
687.         _transferBothExcluded(sender, recipient, amount);
688.     } else {
689.         _transferStandard(sender, recipient, amount);
690.     }
691.
692.     if (!takeFee) restoreAllFee();
693. }
694.
695.     function _transferStandard(address sender, address recipient, uint256 tAmount) private
696.     {
697.         uint256 currentRate = _getRate();
698.         (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
699.         uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
700.         uint256 rBurn = tBurn.mul(currentRate);
701.         uint256 rCharity = tCharity.mul(currentRate);
702.         _standardTransferContent(sender, recipient, rAmount, rTransferAmount);
703.         _sendToCharity(tCharity, sender);
704.         _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
705.         emit Transfer(sender, recipient, tTransferAmount);
706.     }
707.
708.     function _standardTransferContent(address sender, address recipient, uint256 rAmount,
709.     uint256 rTransferAmount) private {
710.         _rOwned[sender] = _rOwned[sender].sub(rAmount);
711.         _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
712.     }
713.
714.     function _transferToExcluded(address sender, address recipient, uint256 tAmount)
715.     private {
716.         uint256 currentRate = _getRate();
717.         (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
718.         uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
719.         uint256 rBurn = tBurn.mul(currentRate);
720.         uint256 rCharity = tCharity.mul(currentRate);
721.         _excludedFromTransferContent(sender, recipient, tTransferAmount, rAmount,
722.         rTransferAmount);
723.         _sendToCharity(tCharity, sender);
724.         _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
725.         emit Transfer(sender, recipient, tTransferAmount);
```



# CyberCrime Shield

cybercrimeshield.org

```
720.     }
721.
722.     function _excludedFromTransferContent(address sender, address recipient, uint256
tTransferAmount, uint256 rAmount, uint256 rTransferAmount) private {
723.         _rOwned[sender] = _rOwned[sender].sub(rAmount);
724.         _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
725.         _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
726.     }
727.
728.
729.     function _transferFromExcluded(address sender, address recipient, uint256 tAmount)
private {
730.         uint256 currentRate = _getRate();
731.         (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
732.         uint256 rBurn = tBurn.mul(currentRate);
733.         uint256 rCharity = tCharity.mul(currentRate);
734.         _excludedToTransferContent(sender, recipient, tAmount, rAmount, rTransferAmount);
735.         _sendToCharity(tCharity, sender);
736.         _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
737.         emit Transfer(sender, recipient, tTransferAmount);
738.     }
739.
740.     function _excludedToTransferContent(address sender, address recipient, uint256
tAmount, uint256 rAmount, uint256 rTransferAmount) private {
741.         _tOwned[sender] = _tOwned[sender].sub(tAmount);
742.         _rOwned[sender] = _rOwned[sender].sub(rAmount);
743.         _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
744.     }
745.
746.     function _transferBothExcluded(address sender, address recipient, uint256 tAmount)
private {
747.         uint256 currentRate = _getRate();
748.         (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
749.         uint256 rBurn = tBurn.mul(currentRate);
750.         uint256 rCharity = tCharity.mul(currentRate);
751.         _bothTransferContent(sender, recipient, tAmount, rAmount, tTransferAmount,
rTransferAmount);
752.         _sendToCharity(tCharity, sender);
753.         _reflectFee(rFee, rBurn, rCharity, tFee, tBurn, tCharity);
754.         emit Transfer(sender, recipient, tTransferAmount);
755.     }
756.
```



# CyberCrime Shield

cybercrimeshield.org

```
757.     function _bothTransferContent(address sender, address recipient, uint256 tAmount,
uint256 rAmount, uint256 tTransferAmount, uint256 rTransferAmount) private {
758.         _tOwned[sender] = _tOwned[sender].sub(tAmount);
759.         _rOwned[sender] = _rOwned[sender].sub(rAmount);
760.         _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
761.         _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
762.     }
763.
764.     function _reflectFee(uint256 rFee, uint256 rBurn, uint256 rCharity, uint256 tFee,
uint256 tBurn, uint256 tCharity) private {
765.         _rTotal = _rTotal.sub(rFee).sub(rBurn).sub(rCharity);
766.         _tFeeTotal = _tFeeTotal.add(tFee);
767.         _tBurnTotal = _tBurnTotal.add(tBurn);
768.         _tCharityTotal = _tCharityTotal.add(tCharity);
769.         _tTotal = _tTotal.sub(tBurn);
770.         emit Transfer(address(this), address(0), tBurn);
771.     }
772.
773.
774.     function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256,
uint256, uint256, uint256, uint256) {
775.         (uint256 tFee, uint256 tBurn, uint256 tCharity) = _getTBasics(tAmount, _TAX_FEE,
_BURN_FEE, _CHARITY_FEE);
776.         uint256 tTransferAmount = getTTransferAmount(tAmount, tFee, tBurn, tCharity);
777.         uint256 currentRate = _getRate();
778.         (uint256 rAmount, uint256 rFee) = _getRBasics(tAmount, tFee, currentRate);
779.         uint256 rTransferAmount = _getRTransferAmount(rAmount, rFee, tBurn, tCharity,
currentRate);
780.         return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tBurn, tCharity);
781.     }
782.
783.     function _getTBasics(uint256 tAmount, uint256 taxFee, uint256 burnFee, uint256
charityFee) private view returns (uint256, uint256, uint256) {
784.         uint256 tFee = ((tAmount.mul(taxFee)).div(_GRANULARITY)).div(100);
785.         uint256 tBurn = ((tAmount.mul(burnFee)).div(_GRANULARITY)).div(100);
786.         uint256 tCharity = ((tAmount.mul(charityFee)).div(_GRANULARITY)).div(100);
787.         return (tFee, tBurn, tCharity);
788.     }
789.
790.     function getTTransferAmount(uint256 tAmount, uint256 tFee, uint256 tBurn, uint256
tCharity) private pure returns (uint256) {
791.         return tAmount.sub(tFee).sub(tBurn).sub(tCharity);
792.     }
793.
```



# CyberCrime Shield

cybercrimeshield.org

```
794.     function _getRBasics(uint256 tAmount, uint256 tFee, uint256 currentRate) private pure
returns (uint256, uint256) {
795.         uint256 rAmount = tAmount.mul(currentRate);
796.         uint256 rFee = tFee.mul(currentRate);
797.         return (rAmount, rFee);
798.     }
799.
800.     function _getRTransferAmount(uint256 rAmount, uint256 rFee, uint256 tBurn, uint256
tCharity, uint256 currentRate) private pure returns (uint256) {
801.         uint256 rBurn = tBurn.mul(currentRate);
802.         uint256 rCharity = tCharity.mul(currentRate);
803.         uint256 rTransferAmount = rAmount.sub(rFee).sub(rBurn).sub(rCharity);
804.         return rTransferAmount;
805.     }
806.
807.     function _getRate() private view returns(uint256) {
808.         (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
809.         return rSupply.div(tSupply);
810.     }
811.
812.     function _getCurrentSupply() private view returns(uint256, uint256) {
813.         uint256 rSupply = _rTotal;
814.         uint256 tSupply = _tTotal;
815.         for (uint256 i = 0; i < _excluded.length; i++) {
816.             if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return
(_rTotal, _tTotal);
817.             rSupply = rSupply.sub(_rOwned[_excluded[i]]);
818.             tSupply = tSupply.sub(_tOwned[_excluded[i]]);
819.         }
820.         if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
821.         return (rSupply, tSupply);
822.     }
823.
824.     function _sendToCharity(uint256 tCharity, address sender) private {
825.         uint256 currentRate = _getRate();
826.         uint256 rCharity = tCharity.mul(currentRate);
827.         _rOwned[FeeAddress] = _rOwned[FeeAddress].add(rCharity);
828.         _tOwned[FeeAddress] = _tOwned[FeeAddress].add(tCharity);
829.         emit Transfer(sender, FeeAddress, tCharity);
830.     }
831.
832.     function removeAllFee() private {
833.         if (_TAX_FEE == 0 && _BURN_FEE == 0 && _CHARITY_FEE == 0) return;
834.
835.         ORIG_TAX_FEE = _TAX_FEE;
```



# CyberCrime Shield

cybercrimeshield.org

```
836.         ORIG_BURN_FEE = _BURN_FEE;
837.         ORIG_CHARITY_FEE = _CHARITY_FEE;
838.
839.         _TAX_FEE = 0;
840.         _BURN_FEE = 0;
841.         _CHARITY_FEE = 0;
842.     }
843.
844.     function restoreAllFee() private {
845.         _TAX_FEE = ORIG_TAX_FEE;
846.         _BURN_FEE = ORIG_BURN_FEE;
847.         _CHARITY_FEE = ORIG_CHARITY_FEE;
848.     }
849.
850.     function _getTaxFee() private view returns(uint256) {
851.         return _TAX_FEE;
852.     }
853.
854.
855. }
```