CyberCrime Shield

cybercrimeshield.org

Smart Contract Audit Report

UNAGI SWAP

https://unagiswap.finance/

AUDIT TYPE: **PUBLIC**

YOU CAN CHECK THE VALIDITY USING THE QR CODE OR LINK:

https://cybercrimeshield.org/secure/unagiswap

Report ID: 291189

July 15, 2021

TABLE OF CONTENTS

SMART CONTRACTS

https://bscscan.com/address/0x7AA509D7761e35bf43B6AeB144480A19d0FEb851#code

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

CRC32: 722C388B

MD5: 530C22ECBE88DB9DB404BCBE612CDD26

SHA-1: 9980FF3C1BFC0F3336D7E1EFB3E1852DEF6FB252

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.

The scope of this audit was to analyze and document the Unagi Swap 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	2
Informational	3

FINDINGS (LOW)

1. Use of assembly

`Inline assembly is a way to access the Ethereum Virtual Machine (which is used in BSC) at a low level. This can discard several important safety features of Solidity.`

line: 390

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

2. View-function should not change state

`In Solidity, functions that do not read from the state or modify it can be declared as` *`view`*

line: 383

```
functionisContract(addressaccount)internalviewreturns(bool){bytes32codehash;bytes32accountHash=0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;assembly{codehash:=extcodehash(account)}return(codehash!=accountHash&&codehash!=0x0);}
```

FINDINGS (INFORMATIONAL)

1. Extra gas consumption

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: 1600

```
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]]);}
```

line: 1655

```
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;}}
```

line: 362

```
x<z
```

CONCLUSION

- Contracts have high code readability
- Gas usage is optimal
- Contracts are fully BSC completable
- No backdoors or overflows are present in the contracts

SOURCE CODE

```
/**
 *Submitted for verification at BscScan.com on 2021-07-11
*/

// SPDX-License-Identifier: MIT

// File: @openzeppelin/contracts/utils/Context.sol

pragma solidity ^0.8.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

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

// File: @openzeppelin/contracts/token/BEP20/IERC20.sol

pragma solidity ^0.8.0;

/**
```

```
37. * @dev Interface of the BEP20 standard as defined in the EIP.
38. */
39.interface IBEP20 {
40.    /**
41.     * @dev Returns the amount of tokens in existence.
42.     */
43.    function totalSupply() external view returns (uint256);
44.
45.    /**
46.     * @dev Returns the amount of tokens owned by `account`.
47.     */
48.    function balanceOf(address account) external view returns (uint256);
49.
50.    /**
51.     * @dev Moves `amount` tokens from the caller's account to
    `recipient`.
52.     *
53.     * Returns a boolean value indicating whether the operation succeeded.
54.     *
55.     * Emits a {Transfer} event.
56.     */
57.    function transfer(address recipient, uint256 amount) external returns
    (bool);
58.
59.    /**
60.     * @dev Returns the remaining number of tokens that `spender` will be
61.     * allowed to spend on behalf of `owner` through {transferFrom}. This
    is
62.     * zero by default.
63.     *
64.     * This value changes when {approve} or {transferFrom} are called.
65.     */
66.    function allowance(address owner, address spender) external view
    returns (uint256);
67.
68.    /**
69.     * @dev Sets `amount` as the allowance of `spender` over the caller's
    tokens.
70.     *
71.     * Returns a boolean value indicating whether the operation succeeded.
72.     *
73.     * IMPORTANT: Beware that changing an allowance with this method
    brings the risk
74.     * that someone may use both the old and the new allowance by
    unfortunate
```

```
75.      * transaction ordering. One possible solution to mitigate this race
76.      * condition is to first reduce the spender's allowance to 0 and set
   the
77.      * desired value afterwards:
78.      * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
79.      *
80.      * Emits an {Approval} event.
81.      */
82.     function approve(address spender, uint256 amount) external returns
   (bool);
83.
84.     /**
85.      * @dev Moves `amount` tokens from `sender` to `recipient` using the
86.      * allowance mechanism. `amount` is then deducted from the caller's
87.      * allowance.
88.      *
89.      * Returns a boolean value indicating whether the operation succeeded.
90.      *
91.      * Emits a {Transfer} event.
92.      */
93.     function transferFrom(address sender, address recipient, uint256
   amount) external returns (bool);
94.
95.     /**
96.      * @dev Emitted when `value` tokens are moved from one account
   (`from`) to
97.      * another (`to`).
98.      *
99.      * Note that `value` may be zero.
100.          */
101.         event Transfer(address indexed from, address indexed to, uint256
   value);
102.
103.         /**
104.          * @dev Emitted when the allowance of a `spender` for an `owner`
   is set by
105.          * a call to {approve}. `value` is the new allowance.
106.          */
107.         event Approval(address indexed owner, address indexed spender,
   uint256 value);
108.     }
109.
110.     // File:
   @openzeppelin/contracts/token/BEP20/extensions/IERC20Metadata.sol
111.
```

```
112.    pragma solidity ^0.8.0;
113.
114.    /**
115.     * @dev Interface for the optional metadata functions from the BEP20 standard.
116.     *
117.     * _Available since v4.1._
118.     */
119.    interface IERC20Metadata is IBEP20 {
120.        /**
121.         * @dev Returns the name of the token.
122.         */
123.        function name() external view returns (string memory);
124.
125.        /**
126.         * @dev Returns the symbol of the token.
127.         */
128.        function symbol() external view returns (string memory);
129.
130.        /**
131.         * @dev Returns the decimals places of the token.
132.         */
133.        function decimals() external view returns (uint8);
134.    }
135.
136.    // File: @openzeppelin/contracts/utils/math/SafeMath.sol
137.
138.    pragma solidity ^0.8.0;
139.
140.    /**
141.     * @dev Wrappers over Solidity's arithmetic operations.
142.     *
143.     * NOTE: `SafeMath` is no longer needed starting with Solidity 0.8. The compiler
144.     * now has built in overflow checking.
145.     */
146.    library SafeMath {
147.        /**
148.         * @dev Returns the addition of two unsigned integers, with an overflow flag.
149.         *
150.         * _Available since v3.4._
151.         */
152.        function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
```

```
153.             unchecked {
154.                 uint256 c = a + b;
155.                 if (c < a) return (false, 0);
156.                 return (true, c);
157.             }
158.         }
159.
160.         /**
161.          * @dev Returns the substraction of two unsigned integers, with
    an overflow flag.
162.          *
163.          * _Available since v3.4._
164.          */
165.         function trySub(uint256 a, uint256 b) internal pure returns
    (bool, uint256) {
166.             unchecked {
167.                 if (b > a) return (false, 0);
168.                 return (true, a - b);
169.             }
170.         }
171.
172.         /**
173.          * @dev Returns the multiplication of two unsigned integers, with
    an overflow flag.
174.          *
175.          * _Available since v3.4._
176.          */
177.         function tryMul(uint256 a, uint256 b) internal pure returns
    (bool, uint256) {
178.             unchecked {
179.                 // Gas optimization: this is cheaper than requiring 'a'
    not being zero, but the
180.                 // benefit is lost if 'b' is also tested.
181.                 // See: https://github.com/OpenZeppelin/openzeppelin-
    contracts/pull/522
182.                 if (a == 0) return (true, 0);
183.                 uint256 c = a * b;
184.                 if (c / a != b) return (false, 0);
185.                 return (true, c);
186.             }
187.         }
188.
189.         /**
190.          * @dev Returns the division of two unsigned integers, with a
    division by zero flag.
```

```
191.          *
192.          * _Available since v3.4._
193.          */
194.         function tryDiv(uint256 a, uint256 b) internal pure returns
    (bool, uint256) {
195.             unchecked {
196.                 if (b == 0) return (false, 0);
197.                 return (true, a / b);
198.             }
199.         }
200.
201.         /**
202.          * @dev Returns the remainder of dividing two unsigned integers,
    with a division by zero flag.
203.          *
204.          * _Available since v3.4._
205.          */
206.         function tryMod(uint256 a, uint256 b) internal pure returns
    (bool, uint256) {
207.             unchecked {
208.                 if (b == 0) return (false, 0);
209.                 return (true, a % b);
210.             }
211.         }
212.
213.         /**
214.          * @dev Returns the addition of two unsigned integers, reverting
    on
215.          * overflow.
216.          *
217.          * Counterpart to Solidity's `+` operator.
218.          *
219.          * Requirements:
220.          *
221.          * - Addition cannot overflow.
222.          */
223.         function add(uint256 a, uint256 b) internal pure returns
    (uint256) {
224.             return a + b;
225.         }
226.
227.         /**
228.          * @dev Returns the subtraction of two unsigned integers,
    reverting on
229.          * overflow (when the result is negative).
```

```
230.          *
231.          * Counterpart to Solidity's `-` operator.
232.          *
233.          * Requirements:
234.          *
235.          * - Subtraction cannot overflow.
236.          */
237.         function sub(uint256 x, uint256 y) internal pure returns (uint256
    z) {
238.             require((z = x - y) <= x);
239.         }
240.
241.         /**
242.          * @dev Returns the multiplication of two unsigned integers,
    reverting on
243.          * overflow.
244.          *
245.          * Counterpart to Solidity's `*` operator.
246.          *
247.          * Requirements:
248.          *
249.          * - Multiplication cannot overflow.
250.          */
251.         function mul(uint256 x, uint256 y) internal pure returns (uint256
    z) {
252.             require(x == 0 || (z = x * y) / x == y);
253.         }
254.
255.         /**
256.          * @dev Returns the integer division of two unsigned integers,
    reverting on
257.          * division by zero. The result is rounded towards zero.
258.          *
259.          * Counterpart to Solidity's `/` operator.
260.          *
261.          * Requirements:
262.          *
263.          * - The divisor cannot be zero.
264.          */
265.         function div(uint256 a, uint256 b) internal pure returns
    (uint256) {
266.             return a / b;
267.         }
268.
269.         /**
```

```
270.          * @dev Returns the remainder of dividing two unsigned integers.
    (unsigned integer modulo),
271.          * reverting when dividing by zero.
272.          *
273.          * Counterpart to Solidity's `%` operator. This function uses a
    `revert`
274.          * opcode (which leaves remaining gas untouched) while Solidity
    uses an
275.          * invalid opcode to revert (consuming all remaining gas).
276.          *
277.          * Requirements:
278.          *
279.          * - The divisor cannot be zero.
280.          */
281.         function mod(uint256 a, uint256 b) internal pure returns
    (uint256) {
282.             return a % b;
283.         }
284.
285.         /**
286.          * @dev Returns the subtraction of two unsigned integers,
    reverting with custom message on
287.          * overflow (when the result is negative).
288.          *
289.          * CAUTION: This function is deprecated because it requires
    allocating memory for the error
290.          * message unnecessarily. For custom revert reasons use {trySub}.
291.          *
292.          * Counterpart to Solidity's `-` operator.
293.          *
294.          * Requirements:
295.          *
296.          * - Subtraction cannot overflow.
297.          */
298.         function sub(
299.             uint256 a,
300.             uint256 b,
301.             string memory errorMessage
302.         ) internal pure returns (uint256) {
303.             unchecked {
304.                 require(b <= a, errorMessage);
305.                 return a - b;
306.             }
307.         }
308.
```

```
309.         /**
310.          * @dev Returns the integer division of two unsigned integers,
   reverting with custom message on
311.          * division by zero. The result is rounded towards zero.
312.          *
313.          * Counterpart to Solidity's `/` operator. Note: this function
   uses a
314.          * `revert` opcode (which leaves remaining gas untouched) while
   Solidity
315.          * uses an invalid opcode to revert (consuming all remaining
   gas).
316.          *
317.          * Requirements:
318.          *
319.          * - The divisor cannot be zero.
320.          */
321.         function div(
322.             uint256 a,
323.             uint256 b,
324.             string memory errorMessage
325.         ) internal pure returns (uint256) {
326.             unchecked {
327.                 require(b > 0, errorMessage);
328.                 return a / b;
329.             }
330.         }
331.
332.         /**
333.          * @dev Returns the remainder of dividing two unsigned integers.
   (unsigned integer modulo),
334.          * reverting with custom message when dividing by zero.
335.          *
336.          * CAUTION: This function is deprecated because it requires
   allocating memory for the error
337.          * message unnecessarily. For custom revert reasons use {tryMod}.
338.          *
339.          * Counterpart to Solidity's `%` operator. This function uses a
   `revert`
340.          * opcode (which leaves remaining gas untouched) while Solidity
   uses an
341.          * invalid opcode to revert (consuming all remaining gas).
342.          *
343.          * Requirements:
344.          *
345.          * - The divisor cannot be zero.
```

```
346.          */
347.         function mod(
348.             uint256 a,
349.             uint256 b,
350.             string memory errorMessage
351.         ) internal pure returns (uint256) {
352.             unchecked {
353.                 require(b > 0, errorMessage);
354.                 return a % b;
355.             }
356.         }
357.
358.         function sqrt(uint y) internal pure returns (uint z) {
359.             if (y > 3) {
360.                 z = y;
361.                 uint x = y / 2 + 1;
362.                 while (x < z) {
363.                     z = x;
364.                     x = (y / x + x) / 2;
365.                 }
366.             } else if (y != 0) {
367.                 z = 1;
368.             }
369.         }
370.
371.         function min(uint x, uint y) internal pure returns (uint z) {
372.             z = x < y ? x : y;
373.         }
374.
375.         function max(uint x, uint y) internal pure returns (uint z) {
376.             z = x > y ? x : y;
377.         }
378.
379.     }
380.
381.     library Address {
382.
383.         function isContract(address account) internal view returns (bool)
    {
384.             // According to EIP-1052, 0x0 is the value returned for not-
    yet created accounts
385.             // and
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
    returned
386.             // for accounts without code, i.e. `keccak256('')`
```

```
387.            bytes32 codehash;
388.            bytes32 accountHash =
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
389.            // solhint-disable-next-line no-inline-assembly
390.            assembly { codehash := extcodehash(account) }
391.            return (codehash != accountHash && codehash != 0x0);
392.        }
393.
394.        function sendValue(address payable recipient, uint256 amount)
    internal {
395.            require(address(this).balance >= amount, "Address:
    insufficient balance");
396.
397.            // solhint-disable-next-line avoid-low-level-calls, avoid-
    call-value
398.            (bool success, ) = recipient.call{ value: amount }("");
399.            require(success, "Address: unable to send value, recipient
    may have reverted");
400.        }
401.
402.
403.        function functionCall(address target, bytes memory data) internal
    returns (bytes memory) {
404.          return functionCall(target, data, "Address: low-level call
    failed");
405.        }
406.
407.        function functionCall(address target, bytes memory data, string
    memory errorMessage) internal returns (bytes memory) {
408.            return _functionCallWithValue(target, data, 0, errorMessage);
409.        }
410.
411.        function functionCallWithValue(address target, bytes memory data,
    uint256 value) internal returns (bytes memory) {
412.            return functionCallWithValue(target, data, value, "Address:
    low-level call with value failed");
413.        }
414.
415.        function functionCallWithValue(address target, bytes memory data,
    uint256 value, string memory errorMessage) internal returns (bytes memory)
    {
416.            require(address(this).balance >= value, "Address:
    insufficient balance for call");
417.            return _functionCallWithValue(target, data, value,
    errorMessage);
```

```
418.         }
419.
420.         function _functionCallWithValue(address target, bytes memory
    data, uint256 weiValue, string memory errorMessage) private returns (bytes
    memory) {
421.             require(isContract(target), "Address: call to non-contract");
422.
423.             (bool success, bytes memory returndata) = target.call{ value:
    weiValue }(data);
424.             if (success) {
425.                 return returndata;
426.             } else {
427.
428.                 if (returndata.length > 0) {
429.                     assembly {
430.                         let returndata_size := mload(returndata)
431.                         revert(add(32, returndata), returndata_size)
432.                     }
433.                 } else {
434.                     revert(errorMessage);
435.                 }
436.             }
437.         }
438.     }
439.
440.     abstract contract Ownable is Context {
441.         address private _owner;
442.         address private _previousOwner;
443.         uint256 private _lockTime;
444.
445.         event OwnershipTransferred(address indexed previousOwner, address
    indexed newOwner);
446.
447.         constructor () {
448.             address msgSender = _msgSender();
449.             _owner = msgSender;
450.             emit OwnershipTransferred(address(0), msgSender);
451.         }
452.
453.         function owner() public view returns (address) {
454.             return _owner;
455.         }
456.
457.         modifier onlyOwner {
```

```
458.             require(_owner == _msgSender(), "Ownable: caller is not the
    owner");
459.             _;
460.         }
461.
462.         function renounceOwnership() public virtual onlyOwner {
463.             emit OwnershipTransferred(_owner, address(0));
464.             _owner = address(0);
465.         }
466.
467.         function transferOwnership(address newOwner) public virtual
    onlyOwner {
468.             require(newOwner != address(0), "Ownable: new owner is the
    zero address");
469.             emit OwnershipTransferred(_owner, newOwner);
470.             _owner = newOwner;
471.         }
472.
473.         function getUnlockTime() public view returns (uint256) {
474.             return _lockTime;
475.         }
476.
477.         function getTime() public view returns (uint256) {
478.             return block.timestamp;
479.         }
480.
481.         function lock(uint256 time) public virtual onlyOwner {
482.             _previousOwner = _owner;
483.             _owner = address(0);
484.             _lockTime = block.timestamp + time;
485.             emit OwnershipTransferred(_owner, address(0));
486.         }
487.
488.         function unlock() public virtual {
489.             require(_previousOwner == msg.sender, "You don't have
    permission to unlock");
490.             require(block.timestamp > _lockTime , "Contract is locked
    until 7 days");
491.             emit OwnershipTransferred(_owner, _previousOwner);
492.             _owner = _previousOwner;
493.         }
494.     }
495.
496.     interface IPancakeFactory {
```

```
497.         event PairCreated(address indexed token0, address indexed token1,
   address pair, uint);
498.
499.         function feeTo() external view returns (address);
500.         function feeToSetter() external view returns (address);
501.
502.         function getPair(address tokenA, address tokenB) external view
   returns (address pair);
503.         function allPairs(uint) external view returns (address pair);
504.         function allPairsLength() external view returns (uint);
505.
506.         function createPair(address tokenA, address tokenB) external
   returns (address pair);
507.
508.         function setFeeTo(address) external;
509.         function setFeeToSetter(address) external;
510.
511.         function INIT_CODE_PAIR_HASH() external view returns (bytes32);
512.     }
513.
514.     interface IPancakePair {
515.         event Approval(address indexed owner, address indexed spender,
   uint value);
516.         event Transfer(address indexed from, address indexed to, uint
   value);
517.
518.         function name() external pure returns (string memory);
519.         function symbol() external pure returns (string memory);
520.         function decimals() external pure returns (uint8);
521.         function totalSupply() external view returns (uint);
522.         function balanceOf(address owner) external view returns (uint);
523.         function allowance(address owner, address spender) external view
   returns (uint);
524.
525.         function approve(address spender, uint value) external returns
   (bool);
526.         function transfer(address to, uint value) external returns
   (bool);
527.         function transferFrom(address from, address to, uint value)
   external returns (bool);
528.
529.         function DOMAIN_SEPARATOR() external view returns (bytes32);
530.         function PERMIT_TYPEHASH() external pure returns (bytes32);
531.         function nonces(address owner) external view returns (uint);
532.
```

```
533.         function permit(address owner, address spender, uint value, uint
   deadline, uint8 v, bytes32 r, bytes32 s) external;
534.
535.         event Mint(address indexed sender, uint amount0, uint amount1);
536.         event Burn(address indexed sender, uint amount0, uint amount1,
   address indexed to);
537.         event Swap(
538.             address indexed sender,
539.             uint amount0In,
540.             uint amount1In,
541.             uint amount0Out,
542.             uint amount1Out,
543.             address indexed to
544.         );
545.         event Sync(uint112 reserve0, uint112 reserve1);
546.
547.         function MINIMUM_LIQUIDITY() external pure returns (uint);
548.         function factory() external view returns (address);
549.         function token0() external view returns (address);
550.         function token1() external view returns (address);
551.         function getReserves() external view returns (uint112 reserve0,
   uint112 reserve1, uint32 blockTimestampLast);
552.         function price0CumulativeLast() external view returns (uint);
553.         function price1CumulativeLast() external view returns (uint);
554.         function kLast() external view returns (uint);
555.
556.         function mint(address to) external returns (uint liquidity);
557.         function burn(address to) external returns (uint amount0, uint
   amount1);
558.         function swap(uint amount0Out, uint amount1Out, address to, bytes
   calldata data) external;
559.         function skim(address to) external;
560.         function sync() external;
561.
562.         function initialize(address, address) external;
563.     }
564.
565.     interface IPancakeRouter01 {
566.         function factory() external pure returns (address);
567.         function WETH() external pure returns (address);
568.
569.         function addLiquidity(
570.             address tokenA,
571.             address tokenB,
572.             uint amountADesired,
```

```
            uint amountBDesired,
            uint amountAMin,
            uint amountBMin,
            address to,
            uint deadline
        ) external returns (uint amountA, uint amountB, uint liquidity);
        function addLiquidityETH(
            address token,
            uint amountTokenDesired,
            uint amountTokenMin,
            uint amountETHMin,
            address to,
            uint deadline
        ) external payable returns (uint amountToken, uint amountETH,
   uint liquidity);
        function removeLiquidity(
            address tokenA,
            address tokenB,
            uint liquidity,
            uint amountAMin,
            uint amountBMin,
            address to,
            uint deadline
        ) external returns (uint amountA, uint amountB);
        function removeLiquidityETH(
            address token,
            uint liquidity,
            uint amountTokenMin,
            uint amountETHMin,
            address to,
            uint deadline
        ) external returns (uint amountToken, uint amountETH);
        function removeLiquidityWithPermit(
            address tokenA,
            address tokenB,
            uint liquidity,
            uint amountAMin,
            uint amountBMin,
            address to,
            uint deadline,
            bool approveMax, uint8 v, bytes32 r, bytes32 s
        ) external returns (uint amountA, uint amountB);
        function removeLiquidityETHWithPermit(
            address token,
            uint liquidity,
```

```
617.             uint amountTokenMin,
618.             uint amountETHMin,
619.             address to,
620.             uint deadline,
621.             bool approveMax, uint8 v, bytes32 r, bytes32 s
622.         ) external returns (uint amountToken, uint amountETH);
623.         function swapExactTokensForTokens(
624.             uint amountIn,
625.             uint amountOutMin,
626.             address[] calldata path,
627.             address to,
628.             uint deadline
629.         ) external returns (uint[] memory amounts);
630.         function swapTokensForExactTokens(
631.             uint amountOut,
632.             uint amountInMax,
633.             address[] calldata path,
634.             address to,
635.             uint deadline
636.         ) external returns (uint[] memory amounts);
637.         function swapExactETHForTokens(uint amountOutMin, address[]
    calldata path, address to, uint deadline)
638.             external
639.             payable
640.             returns (uint[] memory amounts);
641.         function swapTokensForExactETH(uint amountOut, uint amountInMax,
    address[] calldata path, address to, uint deadline)
642.             external
643.             returns (uint[] memory amounts);
644.         function swapExactTokensForETH(uint amountIn, uint amountOutMin,
    address[] calldata path, address to, uint deadline)
645.             external
646.             returns (uint[] memory amounts);
647.         function swapETHForExactTokens(uint amountOut, address[] calldata
    path, address to, uint deadline)
648.             external
649.             payable
650.             returns (uint[] memory amounts);
651.
652.         function quote(uint amountA, uint reserveA, uint reserveB)
    external pure returns (uint amountB);
653.         function getAmountOut(uint amountIn, uint reserveIn, uint
    reserveOut) external pure returns (uint amountOut);
654.         function getAmountIn(uint amountOut, uint reserveIn, uint
    reserveOut) external pure returns (uint amountIn);
```

```
655.         function getAmountsOut(uint amountIn, address[] calldata path)
   external view returns (uint[] memory amounts);
656.         function getAmountsIn(uint amountOut, address[] calldata path)
   external view returns (uint[] memory amounts);
657.     }
658.
659.     interface IPancakeRouter02 is IPancakeRouter01 {
660.         function removeLiquidityETHSupportingFeeOnTransferTokens(
661.             address token,
662.             uint liquidity,
663.             uint amountTokenMin,
664.             uint amountETHMin,
665.             address to,
666.             uint deadline
667.         ) external returns (uint amountETH);
668.         function
   removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
669.             address token,
670.             uint liquidity,
671.             uint amountTokenMin,
672.             uint amountETHMin,
673.             address to,
674.             uint deadline,
675.             bool approveMax, uint8 v, bytes32 r, bytes32 s
676.         ) external returns (uint amountETH);
677.
678.         function swapExactTokensForTokensSupportingFeeOnTransferTokens(
679.             uint amountIn,
680.             uint amountOutMin,
681.             address[] calldata path,
682.             address to,
683.             uint deadline
684.         ) external;
685.         function swapExactETHForTokensSupportingFeeOnTransferTokens(
686.             uint amountOutMin,
687.             address[] calldata path,
688.             address to,
689.             uint deadline
690.         ) external payable;
691.         function swapExactTokensForETHSupportingFeeOnTransferTokens(
692.             uint amountIn,
693.             uint amountOutMin,
694.             address[] calldata path,
695.             address to,
696.             uint deadline
```

```
697.         ) external;
698.     }
699.
700.     // File: @openzeppelin/contracts/security/ReentrancyGuard.sol
701.
702.     pragma solidity ^0.8.0;
703.
704.     /**
705.      * @dev Contract module that helps prevent reentrant calls to a
   function.
706.      *
707.      * Inheriting from `ReentrancyGuard` will make the {nonReentrant}
   modifier
708.      * available, which can be applied to functions to make sure there
   are no nested
709.      * (reentrant) calls to them.
710.      *
711.      * Note that because there is a single `nonReentrant` guard,
   functions marked as
712.      * `nonReentrant` may not call one another. This can be worked around
   by making
713.      * those functions `private`, and then adding `external`
   `nonReentrant` entry
714.      * points to them.
715.      *
716.      * TIP: If you would like to learn more about reentrancy and
   alternative ways
717.      * to protect against it, check out our blog post
718.      * https://blog.openzeppelin.com/reentrancy-after-
   istanbul/[Reentrancy After Istanbul].
719.      */
720.     abstract contract ReentrancyGuard {
721.         // Booleans are more expensive than uint256 or any type that
   takes up a full
722.         // word because each write operation emits an extra SLOAD to
   first read the
723.         // slot's contents, replace the bits taken up by the boolean, and
   then write
724.         // back. This is the compiler's defense against contract upgrades
   and
725.         // pointer aliasing, and it cannot be disabled.
726.
727.         // The values being non-zero value makes deployment a bit more
   expensive,
```

```
728.         // but in exchange the refund on every call to nonReentrant will
   be lower in
729.         // amount. Since refunds are capped to a percentage of the total
730.         // transaction's gas, it is best to keep them low in cases like
   this one, to
731.         // increase the likelihood of the full refund coming into effect.
732.         uint256 private constant _NOT_ENTERED = 1;
733.         uint256 private constant _ENTERED = 2;
734.
735.         uint256 private _status;
736.
737.         constructor() {
738.             _status = _NOT_ENTERED;
739.         }
740.
741.         /**
742.          * @dev Prevents a contract from calling itself, directly or
   indirectly.
743.          * Calling a `nonReentrant` function from another `nonReentrant`
744.          * function is not supported. It is possible to prevent this from
   happening
745.          * by making the `nonReentrant` function external, and make it
   call a
746.          * `private` function that does the actual work.
747.          */
748.         modifier nonReentrant() {
749.             // On the first call to nonReentrant, _notEntered will be
   true
750.             require(_status != _ENTERED, "ReentrancyGuard: reentrant
   call");
751.
752.             // Any calls to nonReentrant after this point will fail
753.             _status = _ENTERED;
754.
755.             _;
756.
757.             // By storing the original value once again, a refund is
   triggered (see
758.             // https://eips.ethereum.org/EIPS/eip-2200)
759.             _status = _NOT_ENTERED;
760.         }
761.         modifier isHuman() {
762.             require(tx.origin == msg.sender, "sorry humans only");
763.             _;
764.         }
```

```
765.     }
766.
767.     // File: @openzeppelin/contracts/token/BEP20/BEP20.sol
768.
769.     pragma solidity ^0.8.0;
770.
771.     /**
772.      * @dev Implementation of the {IERC20} interface.
773.      *
774.      * This implementation is agnostic to the way tokens are created.
   This means
775.      * that a supply mechanism has to be added in a derived contract
   using {_mint}.
776.      * For a generic mechanism see {ERC20PresetMinterPauser}.
777.      *
778.      * TIP: For a detailed writeup see our guide
779.      * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
   mechanisms/226[How
780.      * to implement supply mechanisms].
781.      *
782.      * We have followed general OpenZeppelin guidelines: functions revert
   instead
783.      * of returning `false` on failure. This behavior is nonetheless
   conventional
784.      * and does not conflict with the expectations of BEP20 applications.
785.      *
786.      * Additionally, an {Approval} event is emitted on calls to
   {transferFrom}.
787.      * This allows applications to reconstruct the allowance for all
   accounts just
788.      * by listening to said events. Other implementations of the EIP may
   not emit
789.      * these events, as it isn't required by the specification.
790.      *
791.      * Finally, the non-standard {decreaseAllowance} and
   {increaseAllowance}
792.      * functions have been added to mitigate the well-known issues around
   setting
793.      * allowances. See {IERC20-approve}.
794.      */
795.     contract BEP20 is Context, IBEP20, IERC20Metadata,ReentrancyGuard {
796.         mapping (address => uint256) private _balances;
797.
798.         mapping (address => mapping (address => uint256)) private
   _allowances;
```

```
799.
800.        uint256 private _totalSupply;
801.
802.        string private _name;
803.        string private _symbol;
804.
805.        /**
806.         * @dev Sets the values for {name} and {symbol}.
807.         *
808.         * The defaut value of {decimals} is 18. To select a different
    value for
809.         * {decimals} you should overload it.
810.         *
811.         * All two of these values are immutable: they can only be set
    once during
812.         * construction.
813.         */
814.        constructor (string memory name_, string memory symbol_) {
815.            _name = name_;
816.            _symbol = symbol_;
817.        }
818.
819.        /**
820.         * @dev Returns the name of the token.
821.         */
822.        function name() public view virtual override returns (string
    memory) {
823.            return _name;
824.        }
825.
826.        /**
827.         * @dev Returns the symbol of the token, usually a shorter
    version of the
828.         * name.
829.         */
830.        function symbol() public view virtual override returns (string
    memory) {
831.            return _symbol;
832.        }
833.
834.        /**
835.         * @dev Returns the number of decimals used to get its user
    representation.
836.         * For example, if `decimals` equals `2`, a balance of `505`
    tokens should
```

```
837.          * be displayed to a user as `5,05` (`505 / 10 ** 2`).
838.          *
839.          * Tokens usually opt for a value of 18, imitating the
   relationship between
840.          * Ether and Wei. This is the value {BEP20} uses, unless this
   function is
841.          * overridden;
842.          *
843.          * NOTE: This information is only used for _display_ purposes: it
   in
844.          * no way affects any of the arithmetic of the contract,
   including
845.          * {IERC20-balanceOf} and {IERC20-transfer}.
846.          */
847.         function decimals() public view virtual override returns (uint8)
   {
848.             return 9;
849.         }
850.
851.         /**
852.          * @dev See {IERC20-totalSupply}.
853.          */
854.         function totalSupply() public view virtual override returns
   (uint256) {
855.             return _totalSupply;
856.         }
857.
858.         /**
859.          * @dev See {IERC20-balanceOf}.
860.          */
861.         function balanceOf(address account) public view virtual override
   returns (uint256) {
862.             return _balances[account];
863.         }
864.
865.         /**
866.          * @dev See {IERC20-transfer}.
867.          *
868.          * Requirements:
869.          *
870.          * - `recipient` cannot be the zero address.
871.          * - the caller must have a balance of at least `amount`.
872.          */
873.         function transfer(address recipient, uint256 amount) public
   virtual override nonReentrant returns (bool) {
```

```
874.             _transfer(_msgSender(), recipient, amount);
875.             return true;
876.         }
877.
878.         /**
879.          * @dev See {IERC20-allowance}.
880.          */
881.         function allowance(address owner, address spender) public view
    virtual override returns (uint256) {
882.             return _allowances[owner][spender];
883.         }
884.
885.         /**
886.          * @dev See {IERC20-approve}.
887.          *
888.          * Requirements:
889.          *
890.          * - `spender` cannot be the zero address.
891.          */
892.         function approve(address spender, uint256 amount) public virtual
    override returns (bool) {
893.             _approve(_msgSender(), spender, amount);
894.             return true;
895.         }
896.
897.         /**
898.          * @dev See {IERC20-transferFrom}.
899.          *
900.          * Emits an {Approval} event indicating the updated allowance.
    This is not
901.          * required by the EIP. See the note at the beginning of {BEP20}.
902.          *
903.          * Requirements:
904.          *
905.          * - `sender` and `recipient` cannot be the zero address.
906.          * - `sender` must have a balance of at least `amount`.
907.          * - the caller must have allowance for ``sender``'s tokens of at
    least
908.          * `amount`.
909.          */
910.         function transferFrom(address sender, address recipient, uint256
    amount) public virtual override nonReentrant returns (bool) {
911.             _transfer(sender, recipient, amount);
912.
913.             uint256 currentAllowance = _allowances[sender][_msgSender()];
```

```
914.             require(currentAllowance >= amount, "BEP20: transfer amount
   exceeds allowance");
915.             _approve(sender, _msgSender(), currentAllowance - amount);
916.
917.             return true;
918.         }
919.
920.         /**
921.          * @dev Atomically increases the allowance granted to `spender`
   by the caller.
922.          *
923.          * This is an alternative to {approve} that can be used as a
   mitigation for
924.          * problems described in {IERC20-approve}.
925.          *
926.          * Emits an {Approval} event indicating the updated allowance.
927.          *
928.          * Requirements:
929.          *
930.          * - `spender` cannot be the zero address.
931.          */
932.         function increaseAllowance(address spender, uint256 addedValue)
   public virtual returns (bool) {
933.             _approve(_msgSender(), spender,
   _allowances[_msgSender()][spender] + addedValue);
934.             return true;
935.         }
936.
937.         /**
938.          * @dev Atomically decreases the allowance granted to `spender`
   by the caller.
939.          *
940.          * This is an alternative to {approve} that can be used as a
   mitigation for
941.          * problems described in {IERC20-approve}.
942.          *
943.          * Emits an {Approval} event indicating the updated allowance.
944.          *
945.          * Requirements:
946.          *
947.          * - `spender` cannot be the zero address.
948.          * - `spender` must have allowance for the caller of at least
949.          * `subtractedValue`.
950.          */
```

```
951.        function decreaseAllowance(address spender, uint256
    subtractedValue) public virtual returns (bool) {
952.            uint256 currentAllowance =
    _allowances[_msgSender()][spender];
953.            require(currentAllowance >= subtractedValue, "BEP20:
    decreased allowance below zero");
954.            _approve(_msgSender(), spender, currentAllowance -
    subtractedValue);
955.
956.            return true;
957.        }
958.
959.        /**
960.         * @dev Moves tokens `amount` from `sender` to `recipient`.
961.         *
962.         * This is internal function is equivalent to {transfer}, and can
    be used to
963.         * e.g. implement automatic token fees, slashing mechanisms, etc.
964.         *
965.         * Emits a {Transfer} event.
966.         *
967.         * Requirements:
968.         *
969.         * - `sender` cannot be the zero address.
970.         * - `recipient` cannot be the zero address.
971.         * - `sender` must have a balance of at least `amount`.
972.         */
973.        function _transfer(address sender, address recipient, uint256
    amount) internal virtual {
974.            require(sender != address(0), "BEP20: transfer from the zero
    address");
975.            require(recipient != address(0), "BEP20: transfer to the zero
    address");
976.
977.            _beforeTokenTransfer(sender, recipient, amount);
978.
979.            uint256 senderBalance = _balances[sender];
980.            require(senderBalance >= amount, "BEP20: transfer amount
    exceeds balance");
981.            _balances[sender] = senderBalance - amount;
982.            _balances[recipient] += amount;
983.
984.            emit Transfer(sender, recipient, amount);
985.        }
986.
```

```
987.         /** @dev Creates `amount` tokens and assigns them to `account`,
     increasing
988.          * the total supply.
989.          *
990.          * Emits a {Transfer} event with `from` set to the zero address.
991.          *
992.          * Requirements:
993.          *
994.          * - `to` cannot be the zero address.
995.          */
996.         function _mint(address account, uint256 amount) internal virtual
     {
997.             require(account != address(0), "BEP20: mint to the zero
     address");
998.
999.             _beforeTokenTransfer(address(0), account, amount);
1000.
1001.            _totalSupply += amount;
1002.            _balances[account] += amount;
1003.            emit Transfer(address(0), account, amount);
1004.        }
1005.
1006.        /**
1007.         * @dev Destroys `amount` tokens from `account`, reducing the
1008.         * total supply.
1009.         *
1010.         * Emits a {Transfer} event with `to` set to the zero address.
1011.         *
1012.         * Requirements:
1013.         *
1014.         * - `account` cannot be the zero address.
1015.         * - `account` must have at least `amount` tokens.
1016.         */
1017.        function _burn(address account, uint256 amount) internal virtual
     {
1018.            require(account != address(0), "BEP20: burn from the zero
     address");
1019.
1020.            _beforeTokenTransfer(account, address(0), amount);
1021.
1022.            uint256 accountBalance = _balances[account];
1023.            require(accountBalance >= amount, "BEP20: burn amount exceeds
     balance");
1024.            _balances[account] = accountBalance - amount;
1025.            _totalSupply -= amount;
```

```
1026.
1027.             emit Transfer(account, address(0), amount);
1028.         }
1029.
1030.         /**
1031.          * @dev Sets `amount` as the allowance of `spender` over the
   `owner` s tokens.
1032.          *
1033.          * This internal function is equivalent to `approve`, and can be
   used to
1034.          * e.g. set automatic allowances for certain subsystems, etc.
1035.          *
1036.          * Emits an {Approval} event.
1037.          *
1038.          * Requirements:
1039.          *
1040.          * - `owner` cannot be the zero address.
1041.          * - `spender` cannot be the zero address.
1042.          */
1043.         function _approve(address owner, address spender, uint256 amount)
   internal virtual {
1044.             require(owner != address(0), "BEP20: approve from the zero
   address");
1045.             require(spender != address(0), "BEP20: approve to the zero
   address");
1046.
1047.             _allowances[owner][spender] = amount;
1048.             emit Approval(owner, spender, amount);
1049.         }
1050.
1051.         /**
1052.          * @dev Hook that is called before any transfer of tokens. This
   includes
1053.          * minting and burning.
1054.          *
1055.          * Calling conditions:
1056.          *
1057.          * - when `from` and `to` are both non-zero, `amount` of
   ``from``'s tokens
1058.          * will be to transferred to `to`.
1059.          * - when `from` is zero, `amount` tokens will be minted for
   `to`.
1060.          * - when `to` is zero, `amount` of ``from``'s tokens will be
   burned.
1061.          * - `from` and `to` are never both zero.
```

```
1062.         *
1063.         * To learn more about hooks, head to xref:ROOT:extending-
   contracts.adoc#using-hooks[Using Hooks].
1064.         */
1065.        function _beforeTokenTransfer(address from, address to, uint256
   amount) internal virtual { }
1066.    }
1067.
1068.    // File: contracts/IVaultReferral.sol
1069.
1070.    interface IVaultReferral {
1071.        /**
1072.         * @dev Record referral.
1073.         */
1074.        function recordReferral(address user, address referrer) external;
1075.
1076.        /**
1077.         * @dev Record harvest click count.
1078.         */
1079.        function recordHarvestCount(address user, address referrer)
   external;
1080.
1081.        /**
1082.         * @dev Record referral commission.
1083.         */
1084.        function recordReferralCommission(address referrer, uint256
   commission) external;
1085.
1086.        /**
1087.         * @dev Get the referrer address that referred the user.
1088.         */
1089.        function getReferrer(address user) external view returns
   (address);
1090.        /**
1091.         * @dev Claim Referral Rewards.
1092.         */
1093.        function claim(address user) external view returns (uint256);
1094.    }
1095.
1096.    pragma solidity ^0.8.0;
1097.
1098.    contract UNA is Context, BEP20, Ownable {
1099.        using SafeMath for uint256;
1100.        using Address for address;
1101.
```

```
1102.       address payable public marketingAddress =
    payable(0x0000000000000000000000000000000000000000);
1103.       address payable public jackpotContract =
    payable(0x0000000000000000000000000000000000000000);
1104.       address public immutable deadAddress =
    0x000000000000000000000000000000000000dEaD; //Dead Address
1105.       mapping (address => bool) private operators; // Allowing contract
    as operator to update UNA rewards
1106.       mapping (address => uint256) private _rOwned;
1107.       mapping (address => uint256) private _tOwned;
1108.       mapping (address => mapping (address => uint256)) private
    _allowances;
1109.
1110.       mapping (address => bool) private _isExcludedFromFee;
1111.       mapping(address => bool) private _isExcludedFromMaxTx;
1112.       mapping (address => bool) private _isExcluded;
1113.       address[] private _excluded;
1114.
1115.       uint256 private constant MAX = ~uint256(0);
1116.       uint256 private _tTotal = 1000 * 10**6 * 10**9;
1117.       uint256 private _rTotal = (MAX - (MAX % _tTotal));
1118.       uint256 private _tFeeTotal;
1119.       uint256 private _tBurnTotal;
1120.
1121.       string private _name = "UNA";
1122.       string private _symbol = "UNA";
1123.       uint8 private _decimals = 9;
1124.
1125.       uint256 public _holders = 2;
1126.       uint256 private _previousDcvcHolders = _holders;
1127.
1128.       uint256 public _liquidityFee = 8;
1129.       uint256 private _previousLiquidityFee = _liquidityFee;
1130.
1131.       uint256 public marketingDenominator = 4;
1132.       uint256 public jackpotDenominator = 1;
1133.       uint256 public disruptiveCoverageFee = 2 ether; // AntiWhale
    Solutions
1134.
1135.       uint256 public _maxTxAmount = _tTotal.mul(1).div(1000); // 0.1%
    of total supply
1136.       uint256 private minTokensBeforeSell =
    _maxTxAmount.mul(250).div(100); // Minimum balanceOf(contract) before
    swap, 250% of max transaction amount
1137.
```

```
1138.       uint256 private buyBackCap = 1 * 10**18; // Minimum Buy Back
   Limit 1 BNB or more
1139.       uint256 private bnbToLiquidateTaxes = 1 * 10**6; // Minimum
   address(this).balance
1140.
1141.       mapping (address => uint256) private _purchaseHistory; // Record
   sell history for sell block.timestamp
1142.       uint256 private _rateLimitSeconds = 1; // Antibot trade time
   limit
1143.       uint256 private lastIncreaseSupply;
1144.       uint256 private supplyLimit = 365 days;
1145.
1146.       IPancakeRouter02 public uniswapV2Router;
1147.       address public uniswapV2Pair;
1148.
1149.       IVaultReferral public referral;
1150.
1151.       bool inSwapAndLiquify;
1152.       bool public swapAndLiquifyEnabled = false;
1153.       bool public buyBackEnabled = true;
1154.
1155.       event RewardLiquidityProviders(uint256 amount);
1156.       event BuyBackEnabledUpdated(bool enabled);
1157.       event SwapAndLiquifyEnabledUpdated(bool enabled);
1158.       event SwapAndLiquify(uint256 tokensSwapped,uint256
   bnbReceived,uint256 tokensIntoLiqudity);
1159.       event SwapBNBForTokensBurn(uint256 amountIn,address[] path);
1160.       event SwapTokensForBNB(uint256 amountIn,address[] path);
1161.       event BNBClaimed(address recipient,uint256 bnbReceived,uint256
   nextAvailableClaimDate);
1162.       event OperatorUpdated(address indexed operator, bool indexed
   status);
1163.       event RewardPoolUpdate(address indexed operator, uint256 amount);
1164.
1165.       modifier lockedSwap {
1166.           inSwapAndLiquify = true;
1167.           _;
1168.           inSwapAndLiquify = false;
1169.       }
1170.
1171.       modifier onlyOperate {
1172.           require(operators[msg.sender], "Operator: caller is not the
   operator");
1173.           _;
1174.       }
```

```
1175.
1176.       modifier onlyRegistered {
1177.           require(referral.getReferrer(msg.sender) !=
    address(0),'Sender address not yet registered!');
1178.           _;
1179.       }
1180.
1181.       modifier onlyNewUser {
1182.           require(referral.getReferrer(msg.sender) ==
    address(0),'Sender address had been registered!');
1183.           _;
1184.       }
1185.
1186.       modifier onlyUplineRegistered(address _user) {
1187.           require(referral.getReferrer(_user) != address(0),'Upline
    address not registered!');
1188.           _;
1189.       }
1190.
1191.       constructor () BEP20(_name, _symbol){
1192.           _rOwned[_msgSender()] = _rTotal;
1193.           operators[_msgSender()] = true;
1194.
1195.           /// @dev To exclude contract address, owner address, dead
    address and empty address from max tx limit
1196.           _isExcludedFromMaxTx[owner()] = true;
1197.           _isExcludedFromMaxTx[address(this)] = true;
1198.           _isExcludedFromMaxTx[address(0x000000000000000000000000000000
    000000dEaD)] = true;
1199.           _isExcludedFromMaxTx[address(0)] = true;
1200.
1201.           /// @dev To exclude contract address and owner address from
    fees
1202.           _isExcludedFromFee[owner()] = true;
1203.           _isExcludedFromFee[address(this)] = true;
1204.
1205.           emit Transfer(address(0), _msgSender(), _tTotal);
1206.       }
1207.
1208.       function name() public view override returns (string memory) {
1209.           return _name;
1210.       }
1211.
1212.       function symbol() public view override returns (string memory) {
1213.           return _symbol;
```

```
1214.       }
1215.
1216.       function decimals() public view override returns (uint8) {
1217.           return _decimals;
1218.       }
1219.
1220.       function totalSupply() public view override returns (uint256) {
1221.           return _tTotal;
1222.       }
1223.
1224.       function balanceOf(address account) public view override returns
    (uint256) {
1225.           if (_isExcluded[account]) return _tOwned[account];
1226.           return tokenFromReflection(_rOwned[account]);
1227.       }
1228.
1229.       function transfer(address recipient, uint256 amount) public
    override returns (bool) {
1230.           _transfer(_msgSender(), recipient, amount);
1231.           return true;
1232.       }
1233.
1234.       function allowance(address owner, address spender) public view
    override returns (uint256) {
1235.           return _allowances[owner][spender];
1236.       }
1237.
1238.       function approve(address spender, uint256 amount) public override
    returns (bool) {
1239.           _approve(_msgSender(), spender, amount);
1240.           return true;
1241.       }
1242.
1243.       function transferFrom(address sender, address recipient, uint256
    amount) public override returns (bool) {
1244.           _transfer(sender, recipient, amount);
1245.           _approve(sender, _msgSender(),
    _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer amount
    exceeds allowance"));
1246.           return true;
1247.       }
1248.
1249.       function increaseAllowance(address spender, uint256 addedValue)
    public virtual override returns (bool) {
```

```
1250.           _approve(_msgSender(), spender,
    _allowances[_msgSender()][spender].add(addedValue));
1251.           return true;
1252.       }
1253.
1254.       function decreaseAllowance(address spender, uint256
    subtractedValue) public virtual override returns (bool) {
1255.           _approve(_msgSender(), spender,
    _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20: decreased
    allowance below zero"));
1256.           return true;
1257.       }
1258.
1259.       function isExcludedFromDCVCReward(address account) public view
    returns (bool) {
1260.           return _isExcluded[account];
1261.       }
1262.
1263.       function isExcludedFromFee(address account) public view
    returns(bool) {
1264.           return _isExcludedFromFee[account];
1265.       }
1266.
1267.       function totalFees() public view returns (uint256) {
1268.           return _tFeeTotal;
1269.       }
1270.
1271.       function totalBurn() public view returns (uint256) {
1272.           return _tBurnTotal;
1273.       }
1274.
1275.       function minUNAAmountBeforeSell() public view returns (uint256) {
1276.           return minTokensBeforeSell;
1277.       }
1278.
1279.       function buyBackUNACap() public view returns (uint256) {
1280.           return buyBackCap;
1281.       }
1282.
1283.       function minBNBBuyBack() public view returns (uint256) {
1284.           return bnbToLiquidateTaxes;
1285.       }
1286.
1287.       function sellLimitTimeout() public view returns (uint256) {
1288.           return _rateLimitSeconds;
```

```
1289.       }
1290.
1291.       function lastActivities() public view returns (uint256) {
1292.           return lastIncreaseSupply;
1293.       }
1294.
1295.       function supplyLimitTime() public view returns (uint256) {
1296.           return supplyLimit;
1297.       }
1298.
1299.       function donateFee(uint256 tAmount) public {
1300.           address sender = _msgSender();
1301.           require(!_isExcluded[sender], "Excluded addresses cannot call
   this function");
1302.           (uint256 rAmount,,,,,) = _getValues(tAmount);
1303.           _rOwned[sender] = _rOwned[sender].sub(rAmount);
1304.           _rTotal = _rTotal.sub(rAmount);
1305.           _tFeeTotal = _tFeeTotal.add(tAmount);
1306.       }
1307.
1308.       function reflectionFromToken(uint256 tAmount, bool
   deductTransferFee) public view returns(uint256) {
1309.           require(tAmount <= _tTotal, "Amount must be less than
   supply");
1310.           if (!deductTransferFee) {
1311.               (uint256 rAmount,,,,,) = _getValues(tAmount);
1312.               return rAmount;
1313.           } else {
1314.               (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
1315.               return rTransferAmount;
1316.           }
1317.       }
1318.
1319.       function tokenFromReflection(uint256 rAmount) public view
   returns(uint256) {
1320.           require(rAmount <= _rTotal, "Amount must be less than total
   reflections");
1321.           uint256 currentRate =  _getRate();
1322.           return rAmount.div(currentRate);
1323.       }
1324.
1325.       function _approve(address owner, address spender, uint256 amount)
   internal override {
1326.           require(owner != address(0), "BEP20: approve from the zero
   address");
```

```
1327.           require(spender != address(0), "BEP20: approve to the zero
   address");
1328.
1329.           _allowances[owner][spender] = amount;
1330.           emit Approval(owner, spender, amount);
1331.       }
1332.       /// @dev For user to set their referral
1333.       function setReferral(address _user) public onlyNewUser
   onlyUplineRegistered(_user){
1334.           require(referral.getReferrer(_user) != address(0),'Upline
   address not registered!');
1335.           referral.recordReferral(msg.sender,_user);
1336.       }
1337.
1338.       /// @dev Update lucky ticket via contract, required user balance
   of token bigger than zero
1339.       function luckyTicket(address _user) internal {
1340.           if(balanceOf(_user) > 0){
1341.               referral.recordHarvestCount(_user,referral.getReferrer(_u
   ser));
1342.           }
1343.       }
1344.
1345.       function _transfer(
1346.           address from,
1347.           address to,
1348.           uint256 amount
1349.       ) internal override {
1350.           require(from != address(0), "BEP20: transfer from the zero
   address");
1351.           require(to != address(0), "BEP20: transfer to the zero
   address");
1352.           require(amount > 0, "Transfer amount must be greater than
   zero");
1353.           if(from != owner() && to != owner()) {
1354.               require(amount <= _maxTxAmount, "Transfer amount exceeds
   the maxTxAmount.");
1355.           }
1356.
1357.           checkMaxTxAmount(from,to,amount,msg.value);
1358.           checkBot(from,to);
1359.
1360.           if (!inSwapAndLiquify && swapAndLiquifyEnabled && to ==
   address(uniswapV2Pair)) {
1361.               swapAndLiquifyUNA(from, to);
```

```
            }

            bool takeFee = true;

            /// @dev if any account belongs to _isExcludedFromFee account then remove the fee
            if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
                takeFee = false;
            }

            UNATransfer(from,to,amount,takeFee);
        }

        /// @dev Avoid bot trade
        function checkBot(address sender,address recipient) private {
            if (sender == address(uniswapV2Pair) && recipient != address(0) && recipient != address(uniswapV2Router) && recipient != owner()) {
                    _purchaseHistory[recipient] = block.timestamp;
                }
            if(!_isExcluded[sender] && sender != address(0) &&  sender != address(uniswapV2Pair) && (recipient == address(uniswapV2Pair) || recipient == address(uniswapV2Router))) {
                require(_purchaseHistory[sender].add(_rateLimitSeconds) < block.timestamp, "Error: Are you a bot?");
            }
        }
        /// @dev Check max transaction amount and adding antiwhales
        function checkMaxTxAmount(
            address from,
            address to,
            uint256 amount,
            uint256 value
        ) private view {
            if (
                _isExcludedFromMaxTx[from] == false && // default will be false
                _isExcludedFromMaxTx[to] == false // default will be false
            ) {
                if (value < disruptiveCoverageFee) {
                    require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
                }
            }
```

```
1398.       }
1399.       /// @dev Only sent to jackpot contract and marketing address when
   sell transaction trigger
1400.       function sentJackpotPool(address from, address to,uint256
   initialBalance) private lockedSwap{
1401.           if(from != address(this) && (to !=address(marketingAddress)
   || to !=address(0)) && initialBalance > 0 && (address(marketingAddress) !=
   address(0) && address(jackpotContract) != address(0))){
1402.               uint256 bnbToBeAddedToLiquidity =
   address(this).balance.sub(initialBalance);
1403.
1404.               (bool sent,) = address(marketingAddress).call{value :
   uint256(bnbToBeAddedToLiquidity).mul(marketingDenominator).div(_liquidityF
   ee)}("");
1405.               (bool sent2,) = address(jackpotContract).call{value :
   uint256(bnbToBeAddedToLiquidity).mul(jackpotDenominator).div(_liquidityFee
   )}("");
1406.               if(sent && sent2){
1407.                   emit RewardPoolUpdate(address(this),
   uint256(bnbToBeAddedToLiquidity).mul(marketingDenominator).div(_liquidityF
   ee));
1408.               }
1409.           }
1410.       }
1411.       /// @dev Swap and Liquify
1412.       function swapAndLiquifyUNA(address from, address to) private
   lockedSwap{
1413.           uint256 contractTokenBalance = balanceOf(address(this));
1414.
1415.           if (contractTokenBalance >= _maxTxAmount) {
1416.               contractTokenBalance = _maxTxAmount;
1417.           }
1418.
1419.           bool shouldSell = contractTokenBalance >=
   minTokensBeforeSell;
1420.
1421.           if (
1422.               !inSwapAndLiquify &&
1423.           shouldSell &&
1424.           from != uniswapV2Pair &&
1425.           swapAndLiquifyEnabled &&
1426.           !(from == address(this) && to == address(uniswapV2Pair))
1427.           ) {
1428.               contractTokenBalance = minTokensBeforeSell;
1429.
```

```
1430.                 uint256 pooledBNB = contractTokenBalance.div(2);
1431.                 uint256 piece =
   contractTokenBalance.sub(pooledBNB).div(2);
1432.                 uint256 otherPiece = contractTokenBalance.sub(piece);
1433.
1434.                 uint256 tokenAmountToBeSwapped = pooledBNB.add(piece);
1435.
1436.                 uint256 initialBalance = address(this).balance;
1437.
1438.                 swapTokensForBNB(tokenAmountToBeSwapped);
1439.
1440.                 sentJackpotPool(from,to,initialBalance);
1441.                 uint256 deltaBalance =
   address(this).balance.sub(initialBalance);
1442.
1443.                 uint256 balance = address(this).balance;
1444.                 if (buyBackEnabled && balance >
   uint256(bnbToLiquidateTaxes)) {
1445.
1446.                     if (balance > buyBackCap)
1447.                         balance = buyBackCap;
1448.
1449.                     buyBackUNA(balance.div(100));
1450.                 }
1451.
1452.                 emit SwapAndLiquify(piece, deltaBalance, otherPiece);
1453.             }
1454.         }
1455.         /// @dev Buy back UNA token
1456.         function buyBackUNA(uint256 amount) private lockedSwap {
1457.             if (amount > 0) {
1458.                 swapBNBForTokensBurn(amount);
1459.             }
1460.         }
1461.
1462.         function swapTokensForBNB(uint256 tokenAmount) private {
1463.             address[] memory path = new address[](2);
1464.             path[0] = address(this);
1465.             path[1] = uniswapV2Router.WETH();
1466.
1467.             _approve(address(this), address(uniswapV2Router),
   tokenAmount);
1468.
1469.             uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferT
   okens(
```

```
1470.                 tokenAmount,
1471.                 0, // accept any amount of BNB
1472.                 path,
1473.                 address(this),
1474.                 block.timestamp
1475.             );
1476.
1477.             emit SwapTokensForBNB(tokenAmount, path);
1478.         }
1479.
1480.         function swapBNBForTokensBurn(uint256 amount) private {
1481.             address[] memory path = new address[](2);
1482.             path[0] = uniswapV2Router.WETH();
1483.             path[1] = address(this);
1484.
1485.             uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferT
    okens{value: amount}(
1486.                 0,
1487.                 path,
1488.                 deadAddress,
1489.                 block.timestamp.add(300)
1490.             );
1491.
1492.             emit SwapBNBForTokensBurn(amount, path);
1493.         }
1494.
1495.         /// @dev Token Transfer conditions check
1496.         function UNATransfer(address sender, address recipient, uint256
    amount,bool takeFee) private {
1497.             if(!takeFee)
1498.                 removeAllFee();
1499.
1500.             if (_isExcluded[sender] && !_isExcluded[recipient]) {
1501.                 _transferFromExcluded(sender, recipient, amount);
1502.             } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
1503.                 _transferToExcluded(sender, recipient, amount);
1504.             } else if (_isExcluded[sender] && _isExcluded[recipient]) {
1505.                 _transferBothExcluded(sender, recipient, amount);
1506.             } else {
1507.                 _transferStandard(sender, recipient, amount);
1508.             }
1509.
1510.             if(!takeFee)
1511.                 restoreAllFee();
1512.
```

```
1513.             emit RewardLiquidityProviders(amount);
1514.         }
1515.
1516.         function _transferStandard(address sender, address recipient,
      uint256 tAmount) private {
1517.             (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
      uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
      _getValues(tAmount);
1518.             uint256 currentRate = _getRate();
1519.             _rOwned[sender] = _rOwned[sender].sub(rAmount);
1520.             _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1521.             _takeLiquidity(tLiquidity);
1522.             uint256 rBurn = tLiquidity.mul(currentRate);
1523.             _reflectFee(rFee , rBurn , tFee , tLiquidity);
1524.             emit Transfer(sender, recipient, tTransferAmount);
1525.         }
1526.
1527.         function _transferToExcluded(address sender, address recipient,
      uint256 tAmount) private {
1528.             (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
      uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
      _getValues(tAmount);
1529.             uint256 currentRate = _getRate();
1530.             _rOwned[sender] = _rOwned[sender].sub(rAmount);
1531.             _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
1532.             _rOwned[recipient] =
      _rOwned[recipient].add(rTransferAmount);
1533.             _takeLiquidity(tLiquidity);
1534.             uint256 rBurn = tLiquidity.mul(currentRate);
1535.             _reflectFee(rFee , rBurn , tFee , tLiquidity);
1536.             emit Transfer(sender, recipient, tTransferAmount);
1537.         }
1538.
1539.         function _transferFromExcluded(address sender, address recipient,
      uint256 tAmount) private {
1540.             (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
      uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
      _getValues(tAmount);
1541.             uint256 currentRate = _getRate();
1542.             _tOwned[sender] = _tOwned[sender].sub(tAmount);
1543.             _rOwned[sender] = _rOwned[sender].sub(rAmount);
1544.             _rOwned[recipient] =
      _rOwned[recipient].add(rTransferAmount);
1545.             _takeLiquidity(tLiquidity);
1546.             uint256 rBurn = tLiquidity.mul(currentRate);
```

```
1547.             _reflectFee(rFee , rBurn , tFee , tLiquidity);
1548.             emit Transfer(sender, recipient, tTransferAmount);
1549.         }
1550.
1551.         function _transferBothExcluded(address sender, address recipient,
    uint256 tAmount) private {
1552.             (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
    uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
    _getValues(tAmount);
1553.             uint256 currentRate = _getRate();
1554.             _tOwned[sender] = _tOwned[sender].sub(tAmount);
1555.             _rOwned[sender] = _rOwned[sender].sub(rAmount);
1556.             _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
1557.             _rOwned[recipient] =
    _rOwned[recipient].add(rTransferAmount);
1558.             _takeLiquidity(tLiquidity);
1559.             uint256 rBurn = tLiquidity.mul(currentRate);
1560.             _reflectFee(rFee , rBurn , tFee , tLiquidity);
1561.             emit Transfer(sender, recipient, tTransferAmount);
1562.         }
1563.
1564.         function _reflectFee(uint256 rFee, uint256 rBurn, uint256 tFee,
    uint256 tLiquidity) private {
1565.             _rTotal = _rTotal.sub(rFee).sub(rBurn);
1566.             _tFeeTotal = _tFeeTotal.add(tFee);
1567.             _tBurnTotal = _tBurnTotal.add(tLiquidity);
1568.         }
1569.
1570.         // Private view functions
1571.         function _getValues(uint256 tAmount) private view returns
    (uint256, uint256, uint256, uint256, uint256, uint256) {
1572.             (uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
    _getTValues(tAmount);
1573.             (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) =
    _getRValues(tAmount, tFee, tLiquidity, _getRate());
1574.             return (rAmount, rTransferAmount, rFee, tTransferAmount,
    tFee, tLiquidity);
1575.         }
1576.
1577.         function _getTValues(uint256 tAmount) private view returns
    (uint256, uint256, uint256) {
1578.             uint256 tFee = calculateTaxFee(tAmount);
1579.             uint256 tLiquidity = calculateLiquidityFee(tAmount);
1580.             uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity);
1581.             return (tTransferAmount, tFee, tLiquidity);
```

```
1582.       }
1583.
1584.       function _getRValues(uint256 tAmount, uint256 tFee, uint256
    tLiquidity, uint256 currentRate) private pure returns (uint256, uint256,
    uint256) {
1585.           uint256 rAmount = tAmount.mul(currentRate);
1586.           uint256 rFee = tFee.mul(currentRate);
1587.           uint256 rLiquidity = tLiquidity.mul(currentRate);
1588.           uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);
1589.           return (rAmount, rTransferAmount, rFee);
1590.       }
1591.
1592.       function _getRate() private view returns(uint256) {
1593.           (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
1594.           return rSupply.div(tSupply);
1595.       }
1596.
1597.       function _getCurrentSupply() private view returns(uint256,
    uint256) {
1598.           uint256 rSupply = _rTotal;
1599.           uint256 tSupply = _tTotal;
1600.           for (uint256 i = 0; i < _excluded.length; i++) {
1601.               if (_rOwned[_excluded[i]] > rSupply ||
    _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
1602.               rSupply = rSupply.sub(_rOwned[_excluded[i]]);
1603.               tSupply = tSupply.sub(_tOwned[_excluded[i]]);
1604.           }
1605.           if (rSupply < _rTotal.div(_tTotal)) return (_rTotal,
    _tTotal);
1606.           return (rSupply, tSupply);
1607.       }
1608.
1609.       function _takeLiquidity(uint256 tLiquidity) private {
1610.           uint256 currentRate =  _getRate();
1611.           uint256 rLiquidity = tLiquidity.mul(currentRate);
1612.           _rOwned[address(this)] =
    _rOwned[address(this)].add(rLiquidity);
1613.           if(_isExcluded[address(this)])
1614.               _tOwned[address(this)] =
    _tOwned[address(this)].add(tLiquidity);
1615.       }
1616.
1617.       function calculateTaxFee(uint256 _amount ) private view returns
    (uint256) {
1618.           return _amount.mul(_holders).div(
```

```
1619.                 10**2
1620.             );
1621.         }
1622.
1623.         function calculateLiquidityFee(uint256 _amount) private view
   returns (uint256) {
1624.             return _amount.mul(_liquidityFee).div(
1625.                 10**2
1626.             );
1627.         }
1628.
1629.         function removeAllFee() private {
1630.             if(_holders == 0 && _liquidityFee == 0) return;
1631.
1632.             _previousDcvcHolders = _holders;
1633.             _previousLiquidityFee = _liquidityFee;
1634.
1635.             _holders = 0;
1636.             _liquidityFee = 0;
1637.         }
1638.
1639.         function restoreAllFee() private {
1640.             _holders = _previousDcvcHolders;
1641.             _liquidityFee = _previousLiquidityFee;
1642.         }
1643.         /// @dev For Operator to exclude address in liquidity rewards
1644.         function excludeFromReward(address account) public onlyOperate {
1645.             require(!_isExcluded[account], "Account is already
   excluded");
1646.             if(_rOwned[account] > 0) {
1647.                 _tOwned[account] = tokenFromReflection(_rOwned[account]);
1648.             }
1649.             _isExcluded[account] = true;
1650.             _excluded.push(account);
1651.         }
1652.         /// @dev For Operator to include address in liquidity rewards
1653.         function includeInReward(address account) external onlyOperate {
1654.             require(_isExcluded[account], "Account is already excluded");
1655.             for (uint256 i = 0; i < _excluded.length; i++) {
1656.                 if (_excluded[i] == account) {
1657.                     _excluded[i] = _excluded[_excluded.length - 1];
1658.                     _tOwned[account] = 0;
1659.                     _isExcluded[account] = false;
1660.                     _excluded.pop();
1661.                     break;
```

```
1662.                 }
1663.             }
1664.         }
1665.         /// @dev For Operator to exclude address from fees
1666.         function excludeFromFee(address account) public onlyOperate {
1667.             _isExcludedFromFee[account] = true;
1668.         }
1669.         /// @dev For Operator to include address into fees
1670.         function includeInFee(address account) public onlyOperate {
1671.             _isExcludedFromFee[account] = false;
1672.         }
1673.         /// @dev For Operator to exclude address from max transaction
    amount
1674.         function excludeFromMaxTx(address account) public onlyOperate {
1675.             _isExcludedFromMaxTx[account] = true;
1676.         }
1677.         /// @dev For Operator to include address into max transaction
    amount
1678.         function includeInMaxTx(address account) public onlyOperate {
1679.             _isExcludedFromMaxTx[account] = false;
1680.         }
1681.         /// @dev For Operator to update tax fee
1682.         function setTaxFeePercent(uint256 taxFee) public onlyOperate {
1683.             require(_holders != taxFee && taxFee > 0,'Value unchanged!');
1684.             _holders = taxFee;
1685.         }
1686.         /// @dev For Operator to update liquidity fee
1687.         function setLiquidityFeePercent(uint256 liquidityFee) public
    onlyOperate {
1688.             require(_liquidityFee != liquidityFee && liquidityFee >
    0,'Value unchanged!');
1689.             _liquidityFee = liquidityFee;
1690.         }
1691.         /// @dev For Operator to update maximum transaction amount
1692.         function setMaxTxAmount(uint256 _percentage) public onlyOperate {
1693.             require(_percentage > 0,'Value unchanged!');
1694.             _maxTxAmount = _tTotal.mul(_percentage).div(1000);
1695.         }
1696.         /// @dev For Operator to update jackpot denominator value
1697.         function setJackpotDenominator(uint256 _jackpotDenominator)
    public onlyOperate {
1698.             require(jackpotDenominator != _jackpotDenominator &&
    _jackpotDenominator > 0,'Value unchanged!');
1699.             jackpotDenominator = _jackpotDenominator;
1700.         }
```

```
1701.       /// @dev For Operator to update marketing denominator value
1702.       function setMarketingDenominator(uint256 _marketingDenominator)
   public onlyOperate {
1703.           require(marketingDenominator != _marketingDenominator &&
   _marketingDenominator > 0,'Value unchanged!');
1704.           marketingDenominator = _marketingDenominator;
1705.       }
1706.       /// @dev For Operator to update minimum token balance of contract
   to trigger swap and liquify
1707.       function setNumTokensSellToAddToLiquidity(uint256 _percentage)
   public onlyOperate {
1708.           require(minTokensBeforeSell !=
   _maxTxAmount.mul(_percentage).div(100),'Value unchanged!');
1709.           minTokensBeforeSell = _maxTxAmount.mul(_percentage).div(100);
1710.       }
1711.       /// @dev For Operator to update buy back token limit
1712.       function setBuybackUpperLimit(uint256 _buyBackCap,uint256
   _divider) public onlyOperate {
1713.           require(buyBackCap != _buyBackCap * 10**_divider,'Value
   unchanged!');
1714.           buyBackCap = _buyBackCap * 10**_divider;
1715.       }
1716.       /// @dev For Operator to update minimum BNB balance of contract
   to perform buy back
1717.       function setBNBToLiquidateTaxes(uint256
   _bnbToLiquidateTaxes,uint256 _divider) public onlyOperate {
1718.           require(bnbToLiquidateTaxes != _bnbToLiquidateTaxes *
   10**_divider,'Value unchanged!');
1719.           bnbToLiquidateTaxes = _bnbToLiquidateTaxes * 10**_divider;
1720.       }
1721.       /// @dev For Operator to update marketing address
1722.       function setMarketingAddress(address _marketingAddress) public
   onlyOperate {
1723.           require(marketingAddress != _marketingAddress,'Address
   unchanged!');
1724.           marketingAddress = payable(_marketingAddress);
1725.       }
1726.       /// @dev For Operator to update jackpot contract address
1727.       function setJackpotAddress(address _jackpotContract) public
   onlyOperate {
1728.           require(Address.isContract(_jackpotContract),'Jackpot address
   must be contract!');
1729.           jackpotContract = payable(_jackpotContract);
1730.       }
1731.       /// @dev For operator to enabled swap and liquify
```

```
1732.       function setSwapAndLiquifyEnabled(bool _enabled) public
   onlyOperate {
1733.           swapAndLiquifyEnabled = _enabled;
1734.           emit SwapAndLiquifyEnabledUpdated(_enabled);
1735.       }
1736.       /// @dev For operator to enabled buy back
1737.       function setBuyBackEnabled(bool _enabled) public onlyOperate {
1738.           require(buyBackEnabled != _enabled,'Status unchanged!');
1739.           buyBackEnabled = _enabled;
1740.           emit BuyBackEnabledUpdated(_enabled);
1741.       }
1742.       /// @dev For operator to set interval between each selling action
1743.       function setLimitSellTime(uint256 _sec) public onlyOperate {
1744.           require(_sec >= 1 minutes,'Minimum limit is 1 minute!');
1745.           _rateLimitSeconds = _sec;
1746.       }
1747.       /// @dev For Operator to set Interval between each supply added
1748.       function setSupplyLimitTime(uint256 _supplyLimitsec) public
   onlyOperate {
1749.           require(_supplyLimitsec >= 1 days,'Minimum limit is 1
   days!');
1750.           supplyLimit = _supplyLimitsec;
1751.       }
1752.       /// @dev For Operator to update the status of the operator
1753.       function updateOperator(address _operator, bool _status) external
   onlyOperate {
1754.           require(operators[_operator] != _status,'Status unchanged!');
1755.           operators[_operator] = _status;
1756.           emit OperatorUpdated(_operator, _status);
1757.       }
1758.       /// @dev For Operator to update referral contract address
1759.       function updateReferral(address _referral) public onlyOperate{
1760.           require(Address.isContract(_referral),'Referral address must
   be contract!');
1761.           referral = IVaultReferral(_referral);
1762.       }
1763.       /// @dev Use before Presale
1764.       function activatePresale() external onlyOperate {
1765.           setSwapAndLiquifyEnabled(false);
1766.           _holders = 0;
1767.           _liquidityFee = 0;
1768.           _maxTxAmount = 1000 * 10**6 * 10**9;
1769.       }
1770.       /// @dev Use after Presale ended
1771.       function activateLive() external onlyOperate {
```

```
1772.           setSwapAndLiquifyEnabled(true);
1773.           isExcludedFromFee(address(marketingAddress));
1774.           isExcludedFromFee(address(jackpotContract));
1775.           _holders = 2;
1776.           _liquidityFee = 8;
1777.           setMaxTxAmount(1);
1778.       }
1779.       /// @dev For Operator to initialized Pancake Pair
1780.       function initialized(address _router) public onlyOperate{
1781.           require(uniswapV2Pair == address(0),'Pair already exists!');
1782.           IPancakeRouter02 _uniswapV2Router =
    IPancakeRouter02(_router);
1783.           uniswapV2Pair = IPancakeFactory(_uniswapV2Router.factory())
1784.               .createPair(address(this), _uniswapV2Router.WETH());
1785.
1786.           uniswapV2Router = _uniswapV2Router;
1787.       }
1788.       /// @dev To recieve tokens from uniswapV2Router when swaping
1789.       receive() external payable {}
1790.
1791.       /// @dev Token distribution by Operator only (MAX addresses per
    transaction is 240 addresses)
1792.       function disperseToken(IBEP20 token, address[] memory recipients,
    uint256[] memory values) external onlyOperate{
1793.           uint256 total = 0;
1794.           for (uint256 i = 0; i < recipients.length; i++)
1795.               total += values[i];
1796.           require(token.transferFrom(msg.sender, address(this),
    total));
1797.           for (uint256 i = 0; i < recipients.length; i++)
1798.               require(token.transfer(payable(recipients[i]),
    values[i]));
1799.       }
1800.
1801.       /// @dev Direct token distribution from Operator (Max addresses
    per transaction is 240 addresses)
1802.       function disperseTokenSimple(IBEP20 token, address[] memory
    recipients, uint256[] memory values) external onlyOperate{
1803.           for (uint256 i = 0; i < recipients.length; i++)
1804.               require(token.transferFrom(msg.sender,
    payable(recipients[i]), values[i]));
1805.       }
1806.       /// @dev For operator to deprive other BEP-20 token that was
    mistakenly send to the contract, prohibited from withdraw current contract
    token
```

```
1807.       function rescueTokens(address tokenAddress) external onlyOperate
    {
1808.           IBEP20 token = IBEP20(tokenAddress);
1809.           uint256 tokenBalance = token.balanceOf(address(this));
1810.           require(tokenBalance > 0, "No tokens in our balance");
1811.           require(tokenAddress != address(this),'Prohibited: Cannot
    withdraw contract token!');
1812.           token.transfer(msg.sender, tokenBalance);
1813.       }
1814.
1815.       /// @dev To increase the Supply of token when deadAddress balance
    >= 75% of Total Supply
1816.       function increaseSupply() public onlyOperate {
1817.           require(block.timestamp >=
    lastIncreaseSupply.add(supplyLimit),'Only allow to add supply within
    certain period of time!');
1818.           require(balanceOf(deadAddress) >=
    _tTotal.mul(75).div(100),'UNA token supply unable to increase yet!');
1819.           _tTotal =
    _tTotal.add(balanceOf(deadAddress).mul(1).div(100));
1820.           _rTotal = (MAX - (MAX % _tTotal));
1821.           uint256 newSupply =
    reflectionFromToken(balanceOf(deadAddress).mul(1).div(100),false);
1822.           _rOwned[msg.sender] = _rOwned[msg.sender].add(newSupply);
1823.           lastIncreaseSupply = block.timestamp;
1824.       }
1825.
1826.       /// @dev Disperse Airdrop token to user by Operator (Max
    transaction is 240 addresses)
1827.       function disperseAirdrop(uint256[] memory percent, address[]
    memory _user) public onlyOperate{
1828.           require(percent.length == _user.length,'Different length of
    user and amount!');
1829.           for(uint256 i=0;i<percent.length; i++){
1830.               uint256 airdrop =
    reflectionFromToken(balanceOf(_user[i]).mul(percent[i]).div(100),true);
1831.               _rOwned[msg.sender] = _rOwned[msg.sender].sub(airdrop);
1832.               _rOwned[_user[i]] = _rOwned[_user[i]].add(airdrop);
1833.           }
1834.       }
1835.   }
```