CyberCrime Shield

cybercrimeshield.org

Smart Contract Audit Report

SafeBLAST

https://safeblastcrypto.com

AUDIT TYPE: **PUBLIC**

https://cybercrimeshield.org/secure/safeblast

ID:2390332

May 7, 2021

TABLE OF CONTENTS

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

SMART CONTRACTS

https://bscscan.com/address/0xddc0dbd7dc799ae53a98a60b54999cb6ebb3abf0#code

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

CRC32: 6ECDF5E0

MD5: BCE5D3A1BD3E104376DFCB21BE7AA0C4

SHA-1: 078D18F0227036FFC63CD9E0C983807DE3C7BEBE

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.

SafeBLAST(BLAST) is an Autonomous yield and Liquidity generation protocol. Every time someone buys or sell SafeBLAST token, the total supply goes down and the HODLers get rewarded. Every transaction also creates liquidity, which is automatically LOCKED.

The scope of this audit was to analyze and document the SafeBLAST 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)

Read of persistent state following external call

The contract account state is accessed after an external call to a user defined address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

```
705
706  //exclude owner and this contract from fee
707  _isExcludedFromFee[owner()] = true;
708  _isExcludedFromFee[address(this)] = true;
709
```

Write to persistent state following external call

The contract account state is accessed after an external call to a user defined address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

```
706  //exclude owner and this contract from fee
707  _isExcludedFromFee[owner()] = true;
708  _isExcludedFromFee[address(this)] = true;
709
710  emit Transfer(address(0), _msgSender(), _tTotal);
```

FINDINGS (INFORMATIONAL)

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

```
3   */
4
5   pragma solidity ^0.6.12;
6
    // SPDX-License-Identifier: MIT
```

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

```
698 IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
699 // Create a uniswap pair for this new token
700 uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
701 .createPair(address(this), _uniswapV2Router.WETH());
702
703 // set the rest of the contract variables
```

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

```
699 // Create a uniswap pair for this new token
700 uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
701 .createPair(address(this), _uniswapV2Router.WETH());
702
703 // set the rest of the contract variables
```

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 2021-05-01
*/

pragma solidity ^0.6.12;

// SPDX-License-Identifier: MIT

interface IERC20 {

    function totalSupply() external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
```

```
20.      *
21.      * Returns a boolean value indicating whether the operation succeeded.
22.      *
23.      * Emits a {Transfer} event.
24.      */
25.     function transfer(address recipient, uint256 amount) external returns
    (bool);
26.
27.     /**
28.      * @dev Returns the remaining number of tokens that `spender` will be
29.      * allowed to spend on behalf of `owner` through {transferFrom}. This is
30.      * zero by default.
31.      *
32.      * This value changes when {approve} or {transferFrom} are called.
33.      */
34.     function allowance(address owner, address spender) external view returns
    (uint256);
35.
36.     /**
37.      * @dev Sets `amount` as the allowance of `spender` over the caller's
    tokens.
38.      *
39.      * Returns a boolean value indicating whether the operation succeeded.
40.      *
41.      * IMPORTANT: Beware that changing an allowance with this method brings
    the risk
42.      * that someone may use both the old and the new allowance by unfortunate
43.      * transaction ordering. One possible solution to mitigate this race
44.      * condition is to first reduce the spender's allowance to 0 and set the
45.      * desired value afterwards:
46.      * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
47.      *
48.      * Emits an {Approval} event.
49.      */
50.     function approve(address spender, uint256 amount) external returns (bool);
51.
52.     /**
53.      * @dev Moves `amount` tokens from `sender` to `recipient` using the
54.      * allowance mechanism. `amount` is then deducted from the caller's
55.      * allowance.
56.      *
57.      * Returns a boolean value indicating whether the operation succeeded.
58.      *
59.      * Emits a {Transfer} event.
60.      */
```

```
61.     function transferFrom(address sender, address recipient, uint256 amount)
    external returns (bool);
62. 
63.     /**
64.      * @dev Emitted when `value` tokens are moved from one account (`from`) to
65.      * another (`to`).
66.      *
67.      * Note that `value` may be zero.
68.      */
69.     event Transfer(address indexed from, address indexed to, uint256 value);
70. 
71.     /**
72.      * @dev Emitted when the allowance of a `spender` for an `owner` is set by
73.      * a call to {approve}. `value` is the new allowance.
74.      */
75.     event Approval(address indexed owner, address indexed spender, uint256
    value);
76. }
77. 
78. library SafeMath {
79.     /**
80.      * @dev Returns the addition of two unsigned integers, reverting on
81.      * overflow.
82.      *
83.      * Counterpart to Solidity's `+` operator.
84.      *
85.      * Requirements:
86.      *
87.      * - Addition cannot overflow.
88.      */
89.     function add(uint256 a, uint256 b) internal pure returns (uint256) {
90.         uint256 c = a + b;
91.         require(c >= a, "SafeMath: addition overflow");
92. 
93.         return c;
94.     }
95. 
96.     /**
97.      * @dev Returns the subtraction of two unsigned integers, reverting on
98.      * overflow (when the result is negative).
99.      *
100.          * Counterpart to Solidity's `-` operator.
101.          *
102.          * Requirements:
103.          *
```

```
104.         * - Subtraction cannot overflow.
105.         */
106.        function sub(uint256 a, uint256 b) internal pure returns (uint256) {
107.            return sub(a, b, "SafeMath: subtraction overflow");
108.        }
109.
110.        /**
111.         * @dev Returns the subtraction of two unsigned integers, reverting
    with custom message on
112.         * overflow (when the result is negative).
113.         *
114.         * Counterpart to Solidity's `-` operator.
115.         *
116.         * Requirements:
117.         *
118.         * - Subtraction cannot overflow.
119.         */
120.        function sub(uint256 a, uint256 b, string memory errorMessage) internal
    pure returns (uint256) {
121.            require(b <= a, errorMessage);
122.            uint256 c = a - b;
123.
124.            return c;
125.        }
126.
127.        /**
128.         * @dev Returns the multiplication of two unsigned integers, reverting
    on
129.         * overflow.
130.         *
131.         * Counterpart to Solidity's `*` operator.
132.         *
133.         * Requirements:
134.         *
135.         * - Multiplication cannot overflow.
136.         */
137.        function mul(uint256 a, uint256 b) internal pure returns (uint256) {
138.            // Gas optimization: this is cheaper than requiring 'a' not being
    zero, but the
139.            // benefit is lost if 'b' is also tested.
140.            // See: https://github.com/OpenZeppelin/openzeppelin-
    contracts/pull/522
141.            if (a == 0) {
142.                return 0;
143.            }
```

```
144.
145.             uint256 c = a * b;
146.             require(c / a == b, "SafeMath: multiplication overflow");
147.
148.             return c;
149.         }
150.
151.         /**
152.          * @dev Returns the integer division of two unsigned integers. Reverts
    on
153.          * division by zero. The result is rounded towards zero.
154.          *
155.          * Counterpart to Solidity's `/` operator. Note: this function uses a
156.          * `revert` opcode (which leaves remaining gas untouched) while
    Solidity
157.          * uses an invalid opcode to revert (consuming all remaining gas).
158.          *
159.          * Requirements:
160.          *
161.          * - The divisor cannot be zero.
162.          */
163.         function div(uint256 a, uint256 b) internal pure returns (uint256) {
164.             return div(a, b, "SafeMath: division by zero");
165.         }
166.
167.         /**
168.          * @dev Returns the integer division of two unsigned integers. Reverts
    with custom message on
169.          * division by zero. The result is rounded towards zero.
170.          *
171.          * Counterpart to Solidity's `/` operator. Note: this function uses a
172.          * `revert` opcode (which leaves remaining gas untouched) while
    Solidity
173.          * uses an invalid opcode to revert (consuming all remaining gas).
174.          *
175.          * Requirements:
176.          *
177.          * - The divisor cannot be zero.
178.          */
179.         function div(uint256 a, uint256 b, string memory errorMessage) internal
    pure returns (uint256) {
180.             require(b > 0, errorMessage);
181.             uint256 c = a / b;
182.             // assert(a == b * c + a % b); // There is no case in which this
    doesn't hold
```

```
183.
184.             return c;
185.         }
186.
187.         /**
188.          * @dev Returns the remainder of dividing two unsigned integers.
    (unsigned integer modulo),
189.          * Reverts when dividing by zero.
190.          *
191.          * Counterpart to Solidity's `%` operator. This function uses a
    `revert`
192.          * opcode (which leaves remaining gas untouched) while Solidity uses an
193.          * invalid opcode to revert (consuming all remaining gas).
194.          *
195.          * Requirements:
196.          *
197.          * - The divisor cannot be zero.
198.          */
199.         function mod(uint256 a, uint256 b) internal pure returns (uint256) {
200.             return mod(a, b, "SafeMath: modulo by zero");
201.         }
202.
203.         /**
204.          * @dev Returns the remainder of dividing two unsigned integers.
    (unsigned integer modulo),
205.          * Reverts with custom message when dividing by zero.
206.          *
207.          * Counterpart to Solidity's `%` operator. This function uses a
    `revert`
208.          * opcode (which leaves remaining gas untouched) while Solidity uses an
209.          * invalid opcode to revert (consuming all remaining gas).
210.          *
211.          * Requirements:
212.          *
213.          * - The divisor cannot be zero.
214.          */
215.         function mod(uint256 a, uint256 b, string memory errorMessage) internal
    pure returns (uint256) {
216.             require(b != 0, errorMessage);
217.             return a % b;
218.         }
219.     }
220.
221.     abstract contract Context {
222.         function _msgSender() internal view virtual returns (address payable) {
```

```
223.             return msg.sender;
224.         }
225.
226.         function _msgData() internal view virtual returns (bytes memory) {
227.             this; // silence state mutability warning without generating
    bytecode - see https://github.com/ethereum/solidity/issues/2691
228.             return msg.data;
229.         }
230.     }
231.
232.     library Address {
233.         /**
234.          * @dev Returns true if `account` is a contract.
235.          *
236.          * [IMPORTANT]
237.          * ====
238.          * It is unsafe to assume that an address for which this function
    returns
239.          * false is an externally-owned account (EOA) and not a contract.
240.          *
241.          * Among others, `isContract` will return false for the following
242.          * types of addresses:
243.          *
244.          *  - an externally-owned account
245.          *  - a contract in construction
246.          *  - an address where a contract will be created
247.          *  - an address where a contract lived, but was destroyed
248.          * ====
249.          */
250.         function isContract(address account) internal view returns (bool) {
251.             // According to EIP-1052, 0x0 is the value returned for not-yet
    created accounts
252.             // and
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
253.             // for accounts without code, i.e. `keccak256('')`
254.             bytes32 codehash;
255.             bytes32 accountHash =
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
256.             // solhint-disable-next-line no-inline-assembly
257.             assembly { codehash := extcodehash(account) }
258.             return (codehash != accountHash && codehash != 0x0);
259.         }
260.
261.         /**
262.          * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
```

```
263.         * `recipient`, forwarding all available gas and reverting on errors.
264.         *
265.         * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas
    cost
266.         * of certain opcodes, possibly making contracts go over the 2300 gas
    limit
267.         * imposed by `transfer`, making them unable to receive funds via
268.         * `transfer`. {sendValue} removes this limitation.
269.         *
270.         * https://diligence.consensys.net/posts/2019/09/stop-using-solidity-
    transfer-now/[Learn more].
271.         *
272.         * IMPORTANT: because control is transferred to `recipient`, care must
    be
273.         * taken to not create reentrancy vulnerabilities. Consider using
274.         * {ReentrancyGuard} or the
275.         * https://solidity.readthedocs.io/en/v0.5.11/security-
    considerations.html#use-the-checks-effects-interactions-pattern[checks-
    effects-interactions pattern].
276.         */
277.        function sendValue(address payable recipient, uint256 amount) internal
    {
278.            require(address(this).balance >= amount, "Address: insufficient
    balance");
279.
280.            // solhint-disable-next-line avoid-low-level-calls, avoid-call-
    value
281.            (bool success, ) = recipient.call{ value: amount }("");
282.            require(success, "Address: unable to send value, recipient may have
    reverted");
283.        }
284.
285.        /**
286.         * @dev Performs a Solidity function call using a low level `call`. A
287.         * plain`call` is an unsafe replacement for a function call: use this
288.         * function instead.
289.         *
290.         * If `target` reverts with a revert reason, it is bubbled up by this
291.         * function (like regular Solidity function calls).
292.         *
293.         * Returns the raw returned data. To convert to the expected return
    value,
294.         * use https://solidity.readthedocs.io/en/latest/units-and-global-
    variables.html?highlight=abi.decode#abi-encoding-and-decoding-
    functions[`abi.decode`].
```

```
295.         *
296.         * Requirements:
297.         *
298.         * - `target` must be a contract.
299.         * - calling `target` with `data` must not revert.
300.         *
301.         * _Available since v3.1._
302.         */
303.        function functionCall(address target, bytes memory data) internal
    returns (bytes memory) {
304.            return functionCall(target, data, "Address: low-level call
    failed");
305.        }
306.
307.        /**
308.         * @dev Same as {xref-Address-functionCall-address-bytes-
    }[`functionCall`], but with
309.         * `errorMessage` as a fallback revert reason when `target` reverts.
310.         *
311.         * _Available since v3.1._
312.         */
313.        function functionCall(address target, bytes memory data, string memory
    errorMessage) internal returns (bytes memory) {
314.            return _functionCallWithValue(target, data, 0, errorMessage);
315.        }
316.
317.        /**
318.         * @dev Same as {xref-Address-functionCall-address-bytes-
    }[`functionCall`],
319.         * but also transferring `value` wei to `target`.
320.         *
321.         * Requirements:
322.         *
323.         * - the calling contract must have an ETH balance of at least `value`.
324.         * - the called Solidity function must be `payable`.
325.         *
326.         * _Available since v3.1._
327.         */
328.        function functionCallWithValue(address target, bytes memory data,
    uint256 value) internal returns (bytes memory) {
329.            return functionCallWithValue(target, data, value, "Address: low-
    level call with value failed");
330.        }
331.
332.        /**
```

```
333.         * @dev Same as {xref-Address-functionCallWithValue-address-bytes-
   uint256-}[`functionCallWithValue`], but
334.         * with `errorMessage` as a fallback revert reason when `target`
   reverts.
335.         *
336.         * _Available since v3.1._
337.         */
338.        function functionCallWithValue(address target, bytes memory data,
   uint256 value, string memory errorMessage) internal returns (bytes memory) {
339.            require(address(this).balance >= value, "Address: insufficient
   balance for call");
340.            return _functionCallWithValue(target, data, value, errorMessage);
341.        }
342.
343.        function _functionCallWithValue(address target, bytes memory data,
   uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
344.            require(isContract(target), "Address: call to non-contract");
345.
346.            // solhint-disable-next-line avoid-low-level-calls
347.            (bool success, bytes memory returndata) = target.call{ value:
   weiValue }(data);
348.            if (success) {
349.                return returndata;
350.            } else {
351.                // Look for revert reason and bubble it up if present
352.                if (returndata.length > 0) {
353.                    // The easiest way to bubble the revert reason is using
   memory via assembly
354.
355.                    // solhint-disable-next-line no-inline-assembly
356.                    assembly {
357.                        let returndata_size := mload(returndata)
358.                        revert(add(32, returndata), returndata_size)
359.                    }
360.                } else {
361.                    revert(errorMessage);
362.                }
363.            }
364.        }
365.    }
366.
367.    contract Ownable is Context {
368.        address private _owner;
369.        address private _previousOwner;
370.        uint256 private _lockTime;
```

```
371.
372.       event OwnershipTransferred(address indexed previousOwner, address
   indexed newOwner);
373.
374.       /**
375.        * @dev Initializes the contract setting the deployer as the initial
   owner.
376.        */
377.       constructor () internal {
378.           address msgSender = _msgSender();
379.           _owner = msgSender;
380.           emit OwnershipTransferred(address(0), msgSender);
381.       }
382.
383.       /**
384.        * @dev Returns the address of the current owner.
385.        */
386.       function owner() public view returns (address) {
387.           return _owner;
388.       }
389.
390.       /**
391.        * @dev Throws if called by any account other than the owner.
392.        */
393.       modifier onlyOwner() {
394.           require(_owner == _msgSender(), "Ownable: caller is not the
   owner");
395.           _;
396.       }
397.
398.       /**
399.       * @dev Leaves the contract without owner. It will not be possible to
   call
400.       * `onlyOwner` functions anymore. Can only be called by the current
   owner.
401.       *
402.       * NOTE: Renouncing ownership will leave the contract without an owner,
403.       * thereby removing any functionality that is only available to the
   owner.
404.       */
405.       function renounceOwnership() public virtual onlyOwner {
406.           emit OwnershipTransferred(_owner, address(0));
407.           _owner = address(0);
408.       }
409.
```

```
410.        /**
411.         * @dev Transfers ownership of the contract to a new account
    (`newOwner`).
412.         * Can only be called by the current owner.
413.         */
414.        function transferOwnership(address newOwner) public virtual onlyOwner {
415.            require(newOwner != address(0), "Ownable: new owner is the zero
    address");
416.            emit OwnershipTransferred(_owner, newOwner);
417.            _owner = newOwner;
418.        }
419.
420.        function geUnlockTime() public view returns (uint256) {
421.            return _lockTime;
422.        }
423.
424.        //Locks the contract for owner for the amount of time provided
425.        function lock(uint256 time) public virtual onlyOwner {
426.            _previousOwner = _owner;
427.            _owner = address(0);
428.            _lockTime = now + time;
429.            emit OwnershipTransferred(_owner, address(0));
430.        }
431.
432.        //Unlocks the contract for owner when _lockTime is exceeds
433.        function unlock() public virtual {
434.            require(_previousOwner == msg.sender, "You don't have permission to
    unlock");
435.            require(now > _lockTime , "Contract is locked until 7 days");
436.            emit OwnershipTransferred(_owner, _previousOwner);
437.            _owner = _previousOwner;
438.        }
439.    }
440.
441.    interface IUniswapV2Factory {
442.        event PairCreated(address indexed token0, address indexed token1,
    address pair, uint);
443.
444.        function feeTo() external view returns (address);
445.        function feeToSetter() external view returns (address);
446.
447.        function getPair(address tokenA, address tokenB) external view returns
    (address pair);
448.        function allPairs(uint) external view returns (address pair);
449.        function allPairsLength() external view returns (uint);
```

```
450.
451.       function createPair(address tokenA, address tokenB) external returns
   (address pair);
452.
453.       function setFeeTo(address) external;
454.       function setFeeToSetter(address) external;
455.   }
456.
457.   interface IUniswapV2Pair {
458.       event Approval(address indexed owner, address indexed spender, uint
   value);
459.       event Transfer(address indexed from, address indexed to, uint value);
460.
461.       function name() external pure returns (string memory);
462.       function symbol() external pure returns (string memory);
463.       function decimals() external pure returns (uint8);
464.       function totalSupply() external view returns (uint);
465.       function balanceOf(address owner) external view returns (uint);
466.       function allowance(address owner, address spender) external view
   returns (uint);
467.
468.       function approve(address spender, uint value) external returns (bool);
469.       function transfer(address to, uint value) external returns (bool);
470.       function transferFrom(address from, address to, uint value) external
   returns (bool);
471.
472.       function DOMAIN_SEPARATOR() external view returns (bytes32);
473.       function PERMIT_TYPEHASH() external pure returns (bytes32);
474.       function nonces(address owner) external view returns (uint);
475.
476.       function permit(address owner, address spender, uint value, uint
   deadline, uint8 v, bytes32 r, bytes32 s) external;
477.
478.       event Mint(address indexed sender, uint amount0, uint amount1);
479.       event Burn(address indexed sender, uint amount0, uint amount1, address
   indexed to);
480.       event Swap(
481.           address indexed sender,
482.           uint amount0In,
483.           uint amount1In,
484.           uint amount0Out,
485.           uint amount1Out,
486.           address indexed to
487.       );
488.       event Sync(uint112 reserve0, uint112 reserve1);
```

```
489.
490.        function MINIMUM_LIQUIDITY() external pure returns (uint);
491.        function factory() external view returns (address);
492.        function token0() external view returns (address);
493.        function token1() external view returns (address);
494.        function getReserves() external view returns (uint112 reserve0, uint112
    reserve1, uint32 blockTimestampLast);
495.        function price0CumulativeLast() external view returns (uint);
496.        function price1CumulativeLast() external view returns (uint);
497.        function kLast() external view returns (uint);
498.
499.        function mint(address to) external returns (uint liquidity);
500.        function burn(address to) external returns (uint amount0, uint
    amount1);
501.        function swap(uint amount0Out, uint amount1Out, address to, bytes
    calldata data) external;
502.        function skim(address to) external;
503.        function sync() external;
504.
505.        function initialize(address, address) external;
506.    }
507.
508.    interface IUniswapV2Router01 {
509.        function factory() external pure returns (address);
510.        function WETH() external pure returns (address);
511.
512.        function addLiquidity(
513.            address tokenA,
514.            address tokenB,
515.            uint amountADesired,
516.            uint amountBDesired,
517.            uint amountAMin,
518.            uint amountBMin,
519.            address to,
520.            uint deadline
521.        ) external returns (uint amountA, uint amountB, uint liquidity);
522.        function addLiquidityETH(
523.            address token,
524.            uint amountTokenDesired,
525.            uint amountTokenMin,
526.            uint amountETHMin,
527.            address to,
528.            uint deadline
529.        ) external payable returns (uint amountToken, uint amountETH, uint
    liquidity);
```

```
530.        function removeLiquidity(
531.            address tokenA,
532.            address tokenB,
533.            uint liquidity,
534.            uint amountAMin,
535.            uint amountBMin,
536.            address to,
537.            uint deadline
538.        ) external returns (uint amountA, uint amountB);
539.        function removeLiquidityETH(
540.            address token,
541.            uint liquidity,
542.            uint amountTokenMin,
543.            uint amountETHMin,
544.            address to,
545.            uint deadline
546.        ) external returns (uint amountToken, uint amountETH);
547.        function removeLiquidityWithPermit(
548.            address tokenA,
549.            address tokenB,
550.            uint liquidity,
551.            uint amountAMin,
552.            uint amountBMin,
553.            address to,
554.            uint deadline,
555.            bool approveMax, uint8 v, bytes32 r, bytes32 s
556.        ) external returns (uint amountA, uint amountB);
557.        function removeLiquidityETHWithPermit(
558.            address token,
559.            uint liquidity,
560.            uint amountTokenMin,
561.            uint amountETHMin,
562.            address to,
563.            uint deadline,
564.            bool approveMax, uint8 v, bytes32 r, bytes32 s
565.        ) external returns (uint amountToken, uint amountETH);
566.        function swapExactTokensForTokens(
567.            uint amountIn,
568.            uint amountOutMin,
569.            address[] calldata path,
570.            address to,
571.            uint deadline
572.        ) external returns (uint[] memory amounts);
573.        function swapTokensForExactTokens(
574.            uint amountOut,
```

```
575.            uint amountInMax,
576.            address[] calldata path,
577.            address to,
578.            uint deadline
579.        ) external returns (uint[] memory amounts);
580.        function swapExactETHForTokens(uint amountOutMin, address[] calldata
    path, address to, uint deadline)
581.        external
582.        payable
583.        returns (uint[] memory amounts);
584.        function swapTokensForExactETH(uint amountOut, uint amountInMax,
    address[] calldata path, address to, uint deadline)
585.        external
586.        returns (uint[] memory amounts);
587.        function swapExactTokensForETH(uint amountIn, uint amountOutMin,
    address[] calldata path, address to, uint deadline)
588.        external
589.        returns (uint[] memory amounts);
590.        function swapETHForExactTokens(uint amountOut, address[] calldata path,
    address to, uint deadline)
591.        external
592.        payable
593.        returns (uint[] memory amounts);
594.
595.        function quote(uint amountA, uint reserveA, uint reserveB) external
    pure returns (uint amountB);
596.        function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
    external pure returns (uint amountOut);
597.        function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
    external pure returns (uint amountIn);
598.        function getAmountsOut(uint amountIn, address[] calldata path) external
    view returns (uint[] memory amounts);
599.        function getAmountsIn(uint amountOut, address[] calldata path) external
    view returns (uint[] memory amounts);
600.    }
601.
602.    interface IUniswapV2Router02 is IUniswapV2Router01 {
603.        function removeLiquidityETHSupportingFeeOnTransferTokens(
604.            address token,
605.            uint liquidity,
606.            uint amountTokenMin,
607.            uint amountETHMin,
608.            address to,
609.            uint deadline
610.        ) external returns (uint amountETH);
```

```
611.        function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
612.            address token,
613.            uint liquidity,
614.            uint amountTokenMin,
615.            uint amountETHMin,
616.            address to,
617.            uint deadline,
618.            bool approveMax, uint8 v, bytes32 r, bytes32 s
619.        ) external returns (uint amountETH);
620.
621.        function swapExactTokensForTokensSupportingFeeOnTransferTokens(
622.            uint amountIn,
623.            uint amountOutMin,
624.            address[] calldata path,
625.            address to,
626.            uint deadline
627.        ) external;
628.        function swapExactETHForTokensSupportingFeeOnTransferTokens(
629.            uint amountOutMin,
630.            address[] calldata path,
631.            address to,
632.            uint deadline
633.        ) external payable;
634.        function swapExactTokensForETHSupportingFeeOnTransferTokens(
635.            uint amountIn,
636.            uint amountOutMin,
637.            address[] calldata path,
638.            address to,
639.            uint deadline
640.        ) external;
641.    }
642.
643.    contract BLAST is Context, IERC20, Ownable {
644.        using SafeMath for uint256;
645.        using Address for address;
646.
647.        mapping (address => uint256) private _rOwned;
648.        mapping (address => uint256) private _tOwned;
649.        mapping (address => mapping (address => uint256)) private _allowances;
650.
651.        mapping (address => bool) private _isExcludedFromFee;
652.
653.        mapping (address => bool) private _isExcluded;
654.        address[] private _excluded;
655.
```

```
656.        uint256 private constant MAX = ~uint256(0);
657.        uint256 private _tTotal = 1000000000000000 * 10**9;
658.        uint256 private _rTotal = (MAX - (MAX % _tTotal));
659.        uint256 private _tFeeTotal;
660.
661.        string private _name = "SafeBLAST";
662.        string private _symbol = "BLAST";
663.        uint8 private _decimals = 9;
664.
665.        uint256 public _taxFee = 5;
666.        uint256 private _previousTaxFee = _taxFee;
667.
668.        uint256 public _liquidityFee = 5;
669.        uint256 private _previousLiquidityFee = _liquidityFee;
670.
671.        IUniswapV2Router02 public immutable uniswapV2Router;
672.        address public immutable uniswapV2Pair;
673.
674.        bool inSwapAndLiquify;
675.        bool public swapAndLiquifyEnabled = true;
676.        bool public tradingEnabled = false;
677.
678.        uint256 public _maxTxAmount = 5000000000000 * 10**9;
679.        uint256 private numTokensSellToAddToLiquidity = 500000000000 * 10**9;
680.
681.        event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
682.        event SwapAndLiquifyEnabledUpdated(bool enabled);
683.        event SwapAndLiquify(
684.            uint256 tokensSwapped,
685.            uint256 ethReceived,
686.            uint256 tokensIntoLiqudity
687.        );
688.
689.        modifier lockTheSwap {
690.            inSwapAndLiquify = true;
691.            _;
692.            inSwapAndLiquify = false;
693.        }
694.
695.        constructor () public {
696.            _rOwned[_msgSender()] = _rTotal;
697.
698.            IUniswapV2Router02 _uniswapV2Router =
    IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
699.            // Create a uniswap pair for this new token
```

```
700.            uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
701.            .createPair(address(this), _uniswapV2Router.WETH());
702.
703.            // set the rest of the contract variables
704.            uniswapV2Router = _uniswapV2Router;
705.
706.            //exclude owner and this contract from fee
707.            _isExcludedFromFee[owner()] = true;
708.            _isExcludedFromFee[address(this)] = true;
709.
710.            emit Transfer(address(0), _msgSender(), _tTotal);
711.        }
712.
713.        function name() public view returns (string memory) {
714.            return _name;
715.        }
716.
717.        function symbol() public view returns (string memory) {
718.            return _symbol;
719.        }
720.
721.        function decimals() public view returns (uint8) {
722.            return _decimals;
723.        }
724.
725.        function totalSupply() public view override returns (uint256) {
726.            return _tTotal;
727.        }
728.
729.        function balanceOf(address account) public view override returns
    (uint256) {
730.            if (_isExcluded[account]) return _tOwned[account];
731.            return tokenFromReflection(_rOwned[account]);
732.        }
733.
734.        function transfer(address recipient, uint256 amount) public override
    returns (bool) {
735.            _transfer(_msgSender(), recipient, amount);
736.            return true;
737.        }
738.
739.        function allowance(address owner, address spender) public view override
    returns (uint256) {
740.            return _allowances[owner][spender];
741.        }
```

```
742.
743.       function approve(address spender, uint256 amount) public override
   returns (bool) {
744.           _approve(_msgSender(), spender, amount);
745.           return true;
746.       }
747.
748.       function transferFrom(address sender, address recipient, uint256
   amount) public override returns (bool) {
749.           _transfer(sender, recipient, amount);
750.           _approve(sender, _msgSender(),
   _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds
   allowance"));
751.           return true;
752.       }
753.
754.       function increaseAllowance(address spender, uint256 addedValue) public
   virtual returns (bool) {
755.           _approve(_msgSender(), spender,
   _allowances[_msgSender()][spender].add(addedValue));
756.           return true;
757.       }
758.
759.       function decreaseAllowance(address spender, uint256
   subtractedValue) public virtual returns (bool) {
760.           _approve(_msgSender(), spender,
   _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased
   allowance below zero"));
761.           return true;
762.       }
763.
764.       function isExcludedFromReward(address account) public view returns
   (bool) {
765.           return _isExcluded[account];
766.       }
767.
768.       function totalFees() public view returns (uint256) {
769.           return _tFeeTotal;
770.       }
771.
772.       function deliver(uint256 tAmount) public {
773.           address sender = _msgSender();
774.           require(!_isExcluded[sender], "Excluded addresses cannot call this
   function");
775.           (uint256 rAmount,,,,,) = _getValues(tAmount);
```

```
776.            _rOwned[sender] = _rOwned[sender].sub(rAmount);
777.            _rTotal = _rTotal.sub(rAmount);
778.            _tFeeTotal = _tFeeTotal.add(tAmount);
779.        }
780.
781.        function reflectionFromToken(uint256 tAmount, bool
    deductTransferFee) public view returns(uint256) {
782.            require(tAmount <= _tTotal, "Amount must be less than supply");
783.            if (!deductTransferFee) {
784.                (uint256 rAmount,,,,,) = _getValues(tAmount);
785.                return rAmount;
786.            } else {
787.                (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
788.                return rTransferAmount;
789.            }
790.        }
791.
792.        function tokenFromReflection(uint256 rAmount) public view
    returns(uint256) {
793.            require(rAmount <= _rTotal, "Amount must be less than total
    reflections");
794.            uint256 currentRate =  _getRate();
795.            return rAmount.div(currentRate);
796.        }
797.
798.        function excludeFromReward(address account) public onlyOwner() {
799.            require(!_isExcluded[account], "Account is already excluded");
800.            if(_rOwned[account] > 0) {
801.                _tOwned[account] = tokenFromReflection(_rOwned[account]);
802.            }
803.            _isExcluded[account] = true;
804.            _excluded.push(account);
805.        }
806.
807.        function includeInReward(address account) external onlyOwner() {
808.            require(_isExcluded[account], "Account is already excluded");
809.            for (uint256 i = 0; i < _excluded.length; i++) {
810.                if (_excluded[i] == account) {
811.                    _excluded[i] = _excluded[_excluded.length - 1];
812.                    _tOwned[account] = 0;
813.                    _isExcluded[account] = false;
814.                    _excluded.pop();
815.                    break;
816.                }
817.            }
```

```
818.        }
819.        function _transferBothExcluded(address sender, address recipient,
    uint256 tAmount) private {
820.            (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256
    tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
821.            _tOwned[sender] = _tOwned[sender].sub(tAmount);
822.            _rOwned[sender] = _rOwned[sender].sub(rAmount);
823.            _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
824.            _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
825.            _takeLiquidity(tLiquidity);
826.            _reflectFee(rFee, tFee);
827.            emit Transfer(sender, recipient, tTransferAmount);
828.        }
829.
830.        function excludeFromFee(address account) public onlyOwner {
831.            _isExcludedFromFee[account] = true;
832.        }
833.
834.        function includeInFee(address account) public onlyOwner {
835.            _isExcludedFromFee[account] = false;
836.        }
837.
838.        function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
839.            _taxFee = taxFee;
840.        }
841.
842.        function setLiquidityFeePercent(uint256 liquidityFee) external
    onlyOwner() {
843.            _liquidityFee = liquidityFee;
844.        }
845.
846.        function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
847.            _maxTxAmount = _tTotal.mul(maxTxPercent).div(
848.                10**2
849.            );
850.        }
851.
852.        function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
853.            swapAndLiquifyEnabled = _enabled;
854.            emit SwapAndLiquifyEnabledUpdated(_enabled);
855.        }
856.
857.        function enableTrading() external onlyOwner() {
858.            tradingEnabled = true;
859.        }
```

```
860.
861.        receive() external payable {}
862.
863.        function _reflectFee(uint256 rFee, uint256 tFee) private {
864.            _rTotal = _rTotal.sub(rFee);
865.            _tFeeTotal = _tFeeTotal.add(tFee);
866.        }
867.
868.        function _getValues(uint256 tAmount) private view returns (uint256,
    uint256, uint256, uint256, uint256, uint256) {
869.            (uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
    _getTValues(tAmount);
870.            (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) =
    _getRValues(tAmount, tFee, tLiquidity, _getRate());
871.            return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee,
    tLiquidity);
872.        }
873.
874.        function _getTValues(uint256 tAmount) private view returns (uint256,
    uint256, uint256) {
875.            uint256 tFee = calculateTaxFee(tAmount);
876.            uint256 tLiquidity = calculateLiquidityFee(tAmount);
877.            uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity);
878.            return (tTransferAmount, tFee, tLiquidity);
879.        }
880.
881.        function _getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity,
    uint256 currentRate) private pure returns (uint256, uint256, uint256) {
882.            uint256 rAmount = tAmount.mul(currentRate);
883.            uint256 rFee = tFee.mul(currentRate);
884.            uint256 rLiquidity = tLiquidity.mul(currentRate);
885.            uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);
886.            return (rAmount, rTransferAmount, rFee);
887.        }
888.
889.        function _getRate() private view returns(uint256) {
890.            (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
891.            return rSupply.div(tSupply);
892.        }
893.
894.        function _getCurrentSupply() private view returns(uint256, uint256) {
895.            uint256 rSupply = _rTotal;
896.            uint256 tSupply = _tTotal;
897.            for (uint256 i = 0; i < _excluded.length; i++) {
```

```
898.                if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] >
    tSupply) return (_rTotal, _tTotal);
899.                rSupply = rSupply.sub(_rOwned[_excluded[i]]);
900.                tSupply = tSupply.sub(_tOwned[_excluded[i]]);
901.            }
902.            if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
903.            return (rSupply, tSupply);
904.        }
905.
906.        function _takeLiquidity(uint256 tLiquidity) private {
907.            uint256 currentRate =  _getRate();
908.            uint256 rLiquidity = tLiquidity.mul(currentRate);
909.            _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
910.            if(_isExcluded[address(this)])
911.                _tOwned[address(this)] =
    _tOwned[address(this)].add(tLiquidity);
912.        }
913.
914.        function calculateTaxFee(uint256 _amount) private view returns
    (uint256) {
915.            return _amount.mul(_taxFee).div(
916.                10**2
917.            );
918.        }
919.
920.        function calculateLiquidityFee(uint256 _amount) private view returns
    (uint256) {
921.            return _amount.mul(_liquidityFee).div(
922.                10**2
923.            );
924.        }
925.
926.        function removeAllFee() private {
927.            if(_taxFee == 0 && _liquidityFee == 0) return;
928.
929.            _previousTaxFee = _taxFee;
930.            _previousLiquidityFee = _liquidityFee;
931.
932.            _taxFee = 0;
933.            _liquidityFee = 0;
934.        }
935.
936.        function restoreAllFee() private {
937.            _taxFee = _previousTaxFee;
938.            _liquidityFee = _previousLiquidityFee;
```

```
939.        }
940.
941.        function isExcludedFromFee(address account) public view returns(bool) {
942.            return _isExcludedFromFee[account];
943.        }
944.
945.        function _approve(address owner, address spender, uint256
    amount) private {
946.            require(owner != address(0), "ERC20: approve from the zero
    address");
947.            require(spender != address(0), "ERC20: approve to the zero
    address");
948.
949.            _allowances[owner][spender] = amount;
950.            emit Approval(owner, spender, amount);
951.        }
952.
953.        function _transfer(
954.            address from,
955.            address to,
956.            uint256 amount
957.        ) private {
958.            require(from != address(0), "ERC20: transfer from the zero
    address");
959.            require(to != address(0), "ERC20: transfer to the zero address");
960.            require(amount > 0, "Transfer amount must be greater than zero");
961.
962.            if(from != owner() && to != owner())
963.                require(amount <= _maxTxAmount, "Transfer amount exceeds the
    maxTxAmount.");
964.
965.            if (from != owner() && !tradingEnabled) {
966.                require(tradingEnabled, "Trading is not enabled yet");
967.            }
968.
969.            // is the token balance of this contract address over the min
    number of
970.            // tokens that we need to initiate a swap + liquidity lock?
971.            // also, don't get caught in a circular liquidity event.
972.            // also, don't swap & liquify if sender is uniswap pair.
973.            uint256 contractTokenBalance = balanceOf(address(this));
974.
975.            if(contractTokenBalance >= _maxTxAmount)
976.            {
977.                contractTokenBalance = _maxTxAmount;
```

```
978.            }
979.
980.            bool overMinTokenBalance = contractTokenBalance >=
    numTokensSellToAddToLiquidity;
981.            if (
982.                overMinTokenBalance &&
983.                !inSwapAndLiquify &&
984.                from != uniswapV2Pair &&
985.                swapAndLiquifyEnabled
986.            ) {
987.                contractTokenBalance = numTokensSellToAddToLiquidity;
988.                //add liquidity
989.                swapAndLiquify(contractTokenBalance);
990.            }
991.
992.            //indicates if fee should be deducted from transfer
993.            bool takeFee = true;
994.
995.            //if any account belongs to _isExcludedFromFee account then remove
    the fee
996.            if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
997.                takeFee = false;
998.            }
999.
1000.           //transfer amount, it will take tax, burn, liquidity fee
1001.           _tokenTransfer(from,to,amount,takeFee);
1002.       }
1003.
1004.       function swapAndLiquify(uint256 contractTokenBalance) private
    lockTheSwap {
1005.           // split the contract balance into halves
1006.           uint256 half = contractTokenBalance.div(2);
1007.           uint256 otherHalf = contractTokenBalance.sub(half);
1008.
1009.           // capture the contract's current ETH balance.
1010.           // this is so that we can capture exactly the amount of ETH that
    the
1011.           // swap creates, and not make the liquidity event include any ETH
    that
1012.           // has been manually sent to the contract
1013.           uint256 initialBalance = address(this).balance;
1014.
1015.           // swap tokens for ETH
1016.           swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when
    swap+liquify is triggered
```

```
1017.
1018.           // how much ETH did we just swap into?
1019.           uint256 newBalance = address(this).balance.sub(initialBalance);
1020.
1021.           // add liquidity to uniswap
1022.           addLiquidity(otherHalf, newBalance);
1023.
1024.           emit SwapAndLiquify(half, newBalance, otherHalf);
1025.       }
1026.
1027.       function swapTokensForEth(uint256 tokenAmount) private {
1028.           // generate the uniswap pair path of token -> weth
1029.           address[] memory path = new address[](2);
1030.           path[0] = address(this);
1031.           path[1] = uniswapV2Router.WETH();
1032.
1033.           _approve(address(this), address(uniswapV2Router), tokenAmount);
1034.
1035.           // make the swap
1036.           uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1037.               tokenAmount,
1038.               0, // accept any amount of ETH
1039.               path,
1040.               address(this),
1041.               block.timestamp
1042.           );
1043.       }
1044.
1045.       function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
1046.           // approve token transfer to cover all possible scenarios
1047.           _approve(address(this), address(uniswapV2Router), tokenAmount);
1048.
1049.           // add the liquidity
1050.           uniswapV2Router.addLiquidityETH{value: ethAmount}(
1051.               address(this),
1052.               tokenAmount,
1053.               0, // slippage is unavoidable
1054.               0, // slippage is unavoidable
1055.               owner(),
1056.               block.timestamp
1057.           );
1058.       }
1059.
1060.       //this method is responsible for taking all fee, if takeFee is true
```

```
1061.      function _tokenTransfer(address sender, address recipient, uint256
   amount,bool takeFee) private {
1062.          if(!takeFee)
1063.              removeAllFee();
1064.
1065.          if (_isExcluded[sender] && !_isExcluded[recipient]) {
1066.              _transferFromExcluded(sender, recipient, amount);
1067.          } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
1068.              _transferToExcluded(sender, recipient, amount);
1069.          } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
1070.              _transferStandard(sender, recipient, amount);
1071.          } else if (_isExcluded[sender] && _isExcluded[recipient]) {
1072.              _transferBothExcluded(sender, recipient, amount);
1073.          } else {
1074.              _transferStandard(sender, recipient, amount);
1075.          }
1076.
1077.          if(!takeFee)
1078.              restoreAllFee();
1079.      }
1080.
1081.      function _transferStandard(address sender, address recipient, uint256
   tAmount) private {
1082.          (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256
   tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1083.          _rOwned[sender] = _rOwned[sender].sub(rAmount);
1084.          _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1085.          _takeLiquidity(tLiquidity);
1086.          _reflectFee(rFee, tFee);
1087.          emit Transfer(sender, recipient, tTransferAmount);
1088.      }
1089.
1090.      function _transferToExcluded(address sender, address recipient, uint256
   tAmount) private {
1091.          (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256
   tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1092.          _rOwned[sender] = _rOwned[sender].sub(rAmount);
1093.          _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
1094.          _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1095.          _takeLiquidity(tLiquidity);
1096.          _reflectFee(rFee, tFee);
1097.          emit Transfer(sender, recipient, tTransferAmount);
1098.      }
1099.
```

```
1100.      function _transferFromExcluded(address sender, address recipient,
   uint256 tAmount) private {
1101.          (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256
   tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1102.          _tOwned[sender] = _tOwned[sender].sub(tAmount);
1103.          _rOwned[sender] = _rOwned[sender].sub(rAmount);
1104.          _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1105.          _takeLiquidity(tLiquidity);
1106.          _reflectFee(rFee, tFee);
1107.          emit Transfer(sender, recipient, tTransferAmount);
1108.      }
1109.
1110.
1111.
1112.
1113.  }
```