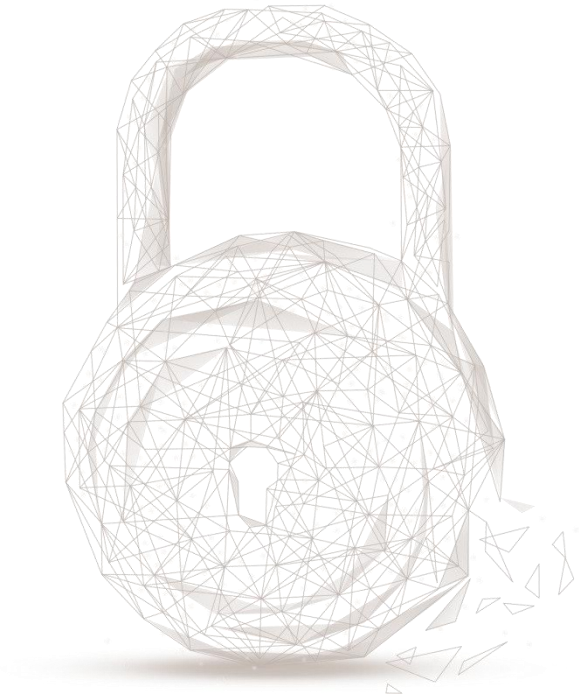
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number: 202106281805

Smart Contract Name:

Lavaswap (Lava)

Smart Contract Address:

0x6e59913074cD836c2904Ddd5b81e85Ec11BD0E02

Smart Contract Address Link:

https://bscscan.com/address/0x6e59913074cd836c2904ddd5b81e85ec11bd0e02#code

Start Date: 2021.06.25

Completion Date: 2021.06.28

Overall Result: Pass (Distinction)

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	BEP-20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass
3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass

		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	N/A
12	Fake Deposit	-	Pass
13	event security	-	Pass

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract Lava, including Coding Standards, Security, and Business Logic. **Lava contract passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1. Basic Token Information

Token name	Lavaswap
Token symbol	Lava
Token decimals	18
Token total supply	Initial 100 million (mintable, the cap is 105 million)
Token type	BEP-20

Table 1 – Basic Token Information

2. Token Vesting Information

N/A

3. Other Functions Description

If the contract's developer address (currently is 0x06e8019f083368febd06f14dab3f1b1487962c1d, can be modified by owner) is not zero address and neither party address to the transfer is a whitelist address, then a certain percentage of the transferring tokens will be destroyed and a part of the transaction fee will be deducted.

```
456   function transfer(address to, uint256 value) public override returns (bool) {
457     require(value <= _balances[msg.sender]);
458     // require(to != address(0));
459
460     uint256 burnValue = 0;
461     uint256 developerValue = 0;
462     uint256 leftValue = value;
463
464      _balances[msg.sender] = _balances[msg.sender].sub(value);
465
466     if(!whiteList[msg.sender] && !whiteList[to] && developerAddr != address(0)){
467         (burnValue,developerValue,leftValue) = afterFee(value);
468     }
469
470     _balances[to] = _balances[to].add(leftValue);
471     _balances[developerAddr] = _balances[developerAddr].add(developerValue);
472     _totalSupply = _totalSupply.sub(burnValue);
473
474     emit Transfer(msg.sender, to, leftValue);
475     emit Transfer(msg.sender, developerAddr, developerValue);
476     emit Transfer(msg.sender, address(0), burnValue);
477
478     return true;
479   }
```

Figure 1 source code of function *transfer*

Therefore, it should be noted that if this token is used to pledge mining and construct trading pairs, please be sure to add the corresponding contract to the whitelist to avoid security vulnerabilities caused by the fact that the amount transferred is not equal to the actual amount received.

Audited Source Code with Comments:

```
/**
 *Submitted for verification at BscScan.com on 2021-06-28
*/

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);
```

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

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

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

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an {Approval} event.
 */
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `sender` to `recipient` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
```

```
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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

// File: @openzeppelin/contracts/math/SafeMath.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

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

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}

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

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

    return c;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `*` operator.
 *
 * Requirements:
 *
 * - Multiplication cannot overflow.
 */
```

```
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
    if (a == 0) {
        return 0;
    }

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

    return c;
}

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

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

    return c;
```

```
    }

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

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

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

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

/*
 * @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 GSN 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 {
    // Beosin (Chengdu LianAn) // Internal function '_msgSender' for getting the caller address.
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }
    // Beosin (Chengdu LianAn) // Internal function '_msgData' for returning the call data.
    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/access/Ownable.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner; // Beosin (Chengdu LianAn) // Declare variable '_owner' for storing the contract
owner.

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); // Beosin
(Chengdu LianAn) // Declare the event 'OwnershipTransferred'.

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender; // Beosin (Chengdu LianAn) // Set the contract deployer as the contract
```

```
owner.
        emit OwnershipTransferred(address(0), msgSender); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return  owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require( owner ==  msgSender(), "Ownable: caller is not the owner"); // Beosin (Chengdu LianAn) // Require that the caller of the modified function must be owner.
         ;
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'newOwner'. Avoid losing ownership.
        emit OwnershipTransferred(_owner, newOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        _owner = newOwner; // Beosin (Chengdu LianAn) // Transfer ownership to 'newOwner'.
    }
}

// File: contracts/Lava.sol
```

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

// import "@nomiclabs/buidler/console.sol";
contract LAVA is IERC20,Ownable {

    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.
    mapping (address => uint256) private  balances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_balances' for storing the token balance of corresponding address.
    mapping (address => mapping (address => uint256)) private  allowed; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowed' for storing the allowance between two addresses.


    string private  name = 'Lavaswap'; // Beosin (Chengdu LianAn) // The token name is 'Lavaswap'.
    string private  symbol = 'Lava'; // Beosin (Chengdu LianAn) // The token symbol is 'Lava'.
    uint8 private _decimals = 18; // Beosin (Chengdu LianAn) // The token decimals is 18.

    // Beosin (Chengdu LianAn) // Return the token name.
    function name() public view returns(string memory) {
        return _name;
    }
    // Beosin (Chengdu LianAn) // Return the token symbol.
    function symbol() public view returns(string memory) {
        return _symbol;
    }
    // Beosin (Chengdu LianAn) // Return the token decimals.
    function decimals() public view returns(uint8) {
        return _decimals;
    }

    uint256 _totalSupply = 0; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' to store the token total supply.
    uint256 _initSupply = 100000000 *1e18; // Beosin (Chengdu LianAn) // The initial total amount of tokens is 100 million.
    uint256 private _cap = 105000000 *1e18; // Beosin (Chengdu LianAn) // The cap of token is 105 million.

    uint256 public burnPercent = 250; // Beosin (Chengdu LianAn) // The initial burning ratio is 2.5%.
    uint256 public maxBurnPercent = 300; // Beosin (Chengdu LianAn) // The maximum burning ratio is 3%.
    uint256 public developerPercent = 250; // Beosin (Chengdu LianAn) // The initial developer allocation ratio is 2.5%.
    uint256 public maxDeveloperPercent = 300; // Beosin (Chengdu LianAn) // The maximum developer allocation ratio is 3%.
    address public developerAddr; // Beosin (Chengdu LianAn) // The developer address.
    mapping (address => bool) public whiteList; // Beosin (Chengdu LianAn) // Declare the variable 'whiteList'
```

```
to store whether the address is in the whitelist.
    mapping (address => bool) public minters; // Beosin (Chengdu LianAn) // Declare the variable 'minters' to
store whether the address has the minting tokens permission.


    // Beosin (Chengdu LianAn) // Grant/revoke minting tokens permission for the specified address.
    function setMinter(address account, bool set) public onlyOwner{
        minters[account] = set;
    }

    modifier onlyMinter() {
        require(minters[msg.sender], "Not minters"); // Beosin (Chengdu LianAn) // Require the caller must has
the minting tokens permission.
            ;
    }
    // Beosin (Chengdu LianAn) // Set developer address.
    function setDev(address   dev) public onlyOwner{
        developerAddr =   dev;
    }
    // Beosin (Chengdu LianAn) // Set the burning ratio.
    function setBurnFee(uint256 _fee) public onlyOwner{
        require(_fee <= maxBurnPercent, "burnPercent exceeded"); // Beosin (Chengdu LianAn) // Require that the
burning ratio should not exceed 3%.
        burnPercent = _fee;
    }

    function setDevFee(uint256 _fee) public onlyOwner{
        require(_fee <= maxDeveloperPercent, "developerPercent exceeded"); // Beosin (Chengdu LianAn) //
Require that the developer allocation ratio should not exceed 3%.
        developerPercent = _fee;
    }
    // Beosin (Chengdu LianAn) // Set the whitelist status of the specified address.
    function setNoFee(address account, bool set) public onlyOwner{
            whiteList[account] = set;
    }

    // Beosin (Chengdu LianAn) // Return the token cap.
    function cap() public view returns (uint256) {
        return _cap;
    }


    // Beosin (Chengdu LianAn) // Constructor, which initializes contract parameters.
    constructor(address _dev) public {
        minters[msg.sender] = true; // Beosin (Chengdu LianAn) // Grant the minting tokens permission to
deployer.
        _mint(msg.sender, _initSupply); // Beosin (Chengdu LianAn) // Mint the initial tokens (100 million) to the
deployer.
```

```
        developerAddr =   dev; // Beosin (Chengdu LianAn) // Set developer address as '_dev'.

    }
    // Beosin (Chengdu LianAn) // Return the token total supply.
    function totalSupply() public view override returns (uint256) {
        return   totalSupply;
    }
    // Beosin (Chengdu LianAn) // Return the token balance of the specified address.
    function balanceOf(address owner) public view override returns (uint256) {
        return   balances[owner];
    }
    // Beosin (Chengdu LianAn) // Return the allowance granted to 'spender' by 'owner'.
    function allowance(address owner, address spender) public view override returns (uint256) {
        return   allowed[owner][spender];
    }

    // Beosin (Chengdu LianAn) // Calculate the number of tokens after the fee is charged.
    function afterFee(uint256 value) public view returns (uint256,uint256,uint256)    {
        uint256 burnValue = value.mul(burnPercent).div(10000); // Beosin (Chengdu LianAn) // The amount of
tokens to burn.
        uint256 developerValue = value.mul(developerPercent).div(10000); // Beosin (Chengdu LianAn) // The
amount of tokens that need to be allocated to the developer.
        uint256 leftValue = value.sub(burnValue).sub(developerValue); // Beosin (Chengdu LianAn) // The amount
of remaining tokens.
        return (burnValue,developerValue,leftValue);
    }
    // Beosin (Chengdu LianAn) // Transfer tokens to the specified address. The user can call this function to
transfer tokens to the zero address.
    function transfer(address to, uint256 value) public override returns (bool) {
        require(value <= _balances[msg.sender]); // Beosin (Chengdu LianAn) // Require the transferring amount
cannot be greater than the caller's token balance.
        // require(to != address(0));

        uint256 burnValue = 0;
        uint256 developerValue = 0;
        uint256 leftValue = value;

         _balances[msg.sender] = _balances[msg.sender].sub(value); // Beosin (Chengdu LianAn) // Update the
token balance of the caller.
        // Beosin (Chengdu LianAn) // If neither the caller nor the destination address is a whitelist address, and
the developer address is not the zero address, the corresponding processing fee is charged.
        if(!whiteList[msg.sender] && !whiteList[to] && developerAddr != address(0)){
            (burnValue,developerValue,leftValue) = afterFee(value); // Beosin (Chengdu LianAn) // Call function
'afterFee' to calculate the amount of burning, allocating and remaining.
        }

        _balances[to] = _balances[to].add(leftValue); // Beosin (Chengdu LianAn) // Update the token balance of
the address 'to'.
```

```
        balances[developerAddr] =   balances[developerAddr].add(developerValue); // Beosin (Chengdu LianAn) // Update the token balance of the devolper.
        _totalSupply = _totalSupply.sub(burnValue); // Beosin (Chengdu LianAn) // Update the token total supply.
        // Beosin (Chengdu LianAn) // Trigger token transfer events.
        emit Transfer(msg.sender, to, leftValue);
        emit Transfer(msg.sender, developerAddr, developerValue);
        emit Transfer(msg.sender, address(0), burnValue);

        return true;
    }

    // Beosin (Chengdu LianAn) // The caller set the allowance granted to 'spender'.
    function approve(address spender, uint256 value) public override returns (bool) {
        require(spender != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.
         allowed[msg.sender][spender] = value; // Beosin (Chengdu LianAn) // The allowance granted to 'spender' by the caller is set to 'value'.
        emit Approval(msg.sender, spender, value); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        return true;
    }
    // Beosin (Chengdu LianAn) // The caller as a delegate of 'from' to transfer the specified amount of tokens to a specified address.
    function transferFrom(address from, address to, uint256 value) public override returns (bool) {
        require(value <= _balances[from]); // Beosin (Chengdu LianAn) // The balance check for 'from'.
        require(value <= _allowed[from][msg.sender]); // Beosin (Chengdu LianAn) // Check the allowance granted to caller by 'from'.
        require(to != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'to'.

        uint256 burnValue = 0;
        uint256 developerValue = 0;
        uint256 leftValue = value;

        _balances[from] = _balances[from].sub(value); // Beosin (Chengdu LianAn) // Update the token balance of 'from'.
        _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value); // Beosin (Chengdu LianAn) // Update the allowance granted to caller by 'from'.
        // Beosin (Chengdu LianAn) // If neither the caller nor the destination address is a whitelist address, and the developer address is not the zero address, the corresponding processing fee is charged.
        if(!whiteList[from] && !whiteList[to] && developerAddr != address(0)){
            (burnValue,developerValue,leftValue) = afterFee(value); // Beosin (Chengdu LianAn) // Call function 'afterFee' to calculate the amount of burning, allocating and remaining.
        }


        _balances[to] = _balances[to].add(leftValue); // Beosin (Chengdu LianAn) // Update the token balance of 'to'.
        _balances[developerAddr] = _balances[developerAddr].add(developerValue); // Beosin (Chengdu LianAn) // Update the token balance of the devolper.
        _totalSupply = _totalSupply.sub(burnValue); // Beosin (Chengdu LianAn) // Update the token total supply.
```

```
        // Beosin (Chengdu LianAn) // Trigger token transfer events.
        emit Transfer(from, to, leftValue);
        emit Transfer(from, developerAddr, developerValue);
        emit Transfer(from, address(0), burnValue);

        return true;
    }
        // Beosin (Chengdu LianAn) // Internal function, mints tokens to the specified address.
        function  mint(address account, uint256 amount) internal virtual {
            require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.
            require(totalSupply().add(amount) <=  cap, "ERC20Capped: cap exceeded"); // Beosin (Chengdu LianAn) // It is required that the total supply of tokens after this minting not exceed the cap.
             totalSupply =  totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the token total supply.
             balances[account] =  balances[account].add(amount); // Beosin (Chengdu LianAn) // Update the token balance of 'account'.
            emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger token transfer event.
        }
        // Beosin (Chengdu LianAn) // Mint tokens to the specified address.
        function mint(address account, uint256 amount) public onlyMinter returns (bool) {
            _mint(account, amount); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to mint tokens.
            return true;
        }
        // Beosin (Chengdu LianAn) // Internal function, destroys the token at the specified address.
        function _burn(address account, uint256 amount) internal virtual {
            require(account != address(0), "ERC20: burn from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.
            _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); // Beosin (Chengdu LianAn) // Update the token balance of 'account'.
            _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the token total supply.
            emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger token transfer event.
        }

        // Beosin (Chengdu LianAn) // Destroy the token at the caller.
        function burn(uint256 amount) public virtual {
            _burn(_msgSender(), amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to destroy tokens.
        }
}
```

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com