Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the CENT on 2022.07.29. The following are the details and results of this smart contract security audit:

Token Name :

CENT

The contract address :

https://scope.klaytn.com/account/0xd364de0683b29e582e5713425b215b24ce804ae9?tabId=contractCode

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Some Risks
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Low Risk

Audit Number : 0X002208030001

Audit Date : 2022.07.29 - 2022.08.03

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that contains the swap section and does not contain dark coin function. The total amount of contract tokens can be changed, users can burn their own tokens through the burn and burnFrom functions. OpenZeppelin's SafeMath security module is used, which is a recommended approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit we found the following information and issues:

1. The owner role can change the minTokensBeforeSwap through the setMinTokensBeforeSwap function.
2. The owner role can update the uniswapV2Router through the updateUniswapV2Router function but the does not check whether the pair has been created in advance. After communication with the project team, they expressed that It was a one time function to call on deployment.
3. The owner role can add or remove users from the isExcludedFromFees list, in this list, users are not charged a fee for transferring transactions.
4. The owner role can change the buyBackContract address through the connectBuyBackContract and the changeBuyBackContract functions.
5. The owner role can set the buyBackFee through the setBuyBackFee function but the buyBackFee can not exceed 15 and it does not record event.

6. The owner role can set the automatedMarketMakerPairs address through the setAutomatedMarketMakerPair function. After communication with the project team, they expressed that the owner set the an new one, and the old one will not be invalid.
7. The swapAndSendFee function has no gas limit when transferring dividends.
8. The swapTokensForEth function does not check the slippage when swapping, if the exchange amount is too large, cent token holders may conduct a sandwich attack. It's recommended to set the minTokensBeforeSwap threshold lower when the price of the token is high.

The source code:

```
// Sources flattened with hardhat v2.8.3 https://hardhat.org

// File @openzeppelin/contracts/utils/Context.sol@v4.4.2

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
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) {
        return msg.data;
    }
}
```

```
// File @openzeppelin/contracts/access/Ownable.sol@v4.4.2
// OpenZeppelin Contracts v4.4.1 (access/Ownable.sol)

pragma solidity ^0.8.0;

/**
 * @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.
 */
abstract contract Ownable is Context {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _transferOwnership(_msgSender());
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }
```

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        //SlowMist// This check is quite good in avoiding losing control of the contract caused by user mistakes
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}

// File @openzeppelin/contracts/security/Pausable.sol@v4.4.2
// OpenZeppelin Contracts v4.4.1 (security/Pausable.sol)

pragma solidity ^0.8.0;

/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
```

```
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
abstract contract Pausable is Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by `account`.
     */
    event Unpaused(address account);

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state.
     */
    constructor() {
        _paused = false;
    }

    /**
     * @dev Returns true if the contract is paused, and false otherwise.
     */
    function paused() public view virtual returns (bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not
paused.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
        require(!paused(), "Pausable: paused");
        _;
    }
```

```
    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    modifier whenPaused() {
        require(paused(), "Pausable: not paused");
        _;
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    function _pause() internal virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function _unpause() internal virtual whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}

// File @openzeppelin/contracts/token/ERC20/IERC20.sol@v4.4.2
// OpenZeppelin Contracts v4.4.1 (token/ERC20/IERC20.sol)

pragma solidity ^0.8.0;

/**
 * @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/token/ERC20/extensions/IERC20Metadata.sol@v4.4.2
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/IERC20Metadata.sol)

pragma solidity ^0.8.0;

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
```

```
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}

// File @openzeppelin/contracts/token/ERC20/ERC20.sol@v4.4.2
// OpenZeppelin Contracts v4.4.1 (token/ERC20/ERC20.sol)

pragma solidity ^0.8.0;

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
```

```
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
```

```
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5.05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless this function is
     * overridden;
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual override returns (uint8) {
        return 18;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtual override returns
(uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override
returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }
```

```
    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override
returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns
(bool) {
        _approve(_msgSender(), spender, amount);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);

        uint256 currentAllowance = _allowances[sender][_msgSender()];
```

```
        require(currentAllowance >= amount, "ERC20: transfer amount exceeds
allowance");
        unchecked {
            _approve(sender, _msgSender(), currentAllowance - amount);
        }
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual
returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender] +
addedValue);
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public
virtual returns (bool) {
```

```
        uint256 currentAllowance = _allowances[_msgSender()][spender];
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance
below zero");
        unchecked {
            _approve(_msgSender(), spender, currentAllowance - subtractedValue);
        }

        return true;
    }

    /**
     * @dev Moves `amount` of tokens from `sender` to `recipient`.
     *
     * This internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = _balances[sender];
        require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[sender] = senderBalance - amount;
        }
        _balances[recipient] += amount;

        emit Transfer(sender, recipient, amount);

        _afterTokenTransfer(sender, recipient, amount);
```

```
}

/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}

/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    uint256 accountBalance = _balances[account];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    unchecked {
        _balances[account] = accountBalance - amount;
    }
```

```
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
```

```
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-
hooks[Using Hooks].
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}

    /**
     * @dev Hook that is called after any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-
hooks[Using Hooks].
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}

// File @openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol@v4.4.2
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/ERC20Burnable.sol)

pragma solidity ^0.8.0;

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
```

```
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    //SlowMist// Because burnFrom() and transferFrom() share the allowed amount of
approve(), if the agent be evil, there is the possibility of malicious burn
    function burnFrom(address account, uint256 amount) public virtual {
        uint256 currentAllowance = allowance(account, _msgSender());
        require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
        unchecked {
            _approve(account, _msgSender(), currentAllowance - amount);
        }
        _burn(account, amount);
    }
}

// File @openzeppelin/contracts/utils/math/SafeMath.sol@v4.4.2
// OpenZeppelin Contracts v4.4.1 (utils/math/SafeMath.sol)

pragma solidity ^0.8.0;

// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.

/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is generally not needed starting with Solidity 0.8, since the
compiler
```

```
 * now has built in overflow checking.
 */
//SlowMist// OpenZeppelin's SafeMath security module is used, which is a recommended approach
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            // 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 (true, 0);
            uint256 c = a * b;
```

```
            if (c / a != b) return (false, 0);
            return (true, c);
        }
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero
flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division
by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }

    /**
     * @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) {
        return a + b;
    }
```

```
/**
 * @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 a - b;
}

/**
 * @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) {
    return a * b;
}

/**
 * @dev Returns the integer division of two unsigned integers, reverting on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator.
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b) internal pure returns (uint256) {
    return a / b;
}

/**
```

```
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting 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 a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom
message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for
the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b <= a, errorMessage);
            return a - b;
        }
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting 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) {
        unchecked {
            require(b > 0, errorMessage);
            return a / b;
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for
the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * 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) {
```

```
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}

pragma solidity >=0.5.0;

interface IUniswapV2Factory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    function migrator() external view returns (address);

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function feeDistributor() external view returns (address);

    function klayBuybackFund() external view returns (address);

    function getPair(address tokenA, address tokenB)
        external
        view
        returns (address pair);

    function allPairs(uint256) external view returns (address pair);

    function allPairsLength() external view returns (uint256);

    function createPair(address tokenA, address tokenB)
        external
        returns (address pair);

    function createPair(address tokenA, address tokenB, uint8 decimals)
        external
        returns (address pair);
```

```
    function setFeeTo(address) external;

    function setFeeToSetter(address) external;

    function setFeeDistributor(address) external;

    function setKlayBuybackFund(address) external;

    function setMigrator(address) external;
}

pragma solidity >=0.5.0;

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns
(bool);

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed
to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
```

```
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1,
uint32 blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data)
external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

pragma solidity >=0.6.2;

interface IUniswapV2Router02 {
    // 01
    function factory() external pure returns (address);
    function WKLAY() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityKLAY(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountKLAYMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountKLAY, 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 removeLiquidityKLAY(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountKLAYMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountKLAY);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactKLAYForTokens(uint amountOutMin, address[] calldata path,
address to, uint deadline)
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactKLAY(uint amountOut, uint amountInMax, address[]
calldata path, address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapExactTokensForKLAY(uint amountIn, uint amountOutMin, address[]
calldata path, address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapKLAYForExactTokens(uint amountOut, address[] calldata path, address
```

```
to, uint deadline)
        external
        payable
        returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns
(uint amountB);
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external
pure returns (uint amountOut);
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external
pure returns (uint amountIn);
    function getAmountsOut(uint amountIn, address[] calldata path) external view
returns (uint[] memory amounts);
    function getAmountsIn(uint amountOut, address[] calldata path) external view
returns (uint[] memory amounts);

    // 02
    function removeLiquidityKLAYSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountKLAYMin,
        address to,
        uint deadline
    ) external returns (uint amountKLAY);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactKLAYForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForKLAYSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
```

```
        uint deadline
    ) external;
}

// File contracts/CENT.sol

pragma solidity 0.8.7;

contract ERC20DividendToken is Context, ERC20Burnable, Pausable, Ownable {
    using SafeMath for uint256;

    IUniswapV2Router02 public uniswapV2Router;
    address public uniswapV2Pair;

    bool private swapping;

    uint256 public minTokensBeforeSwap;

    uint256 public buyBackFee;
    address public buyBackContract;

    bool public buyBackContractConnected;

    mapping(address => bool) public isExcludedFromFees;
    mapping(address => bool) public automatedMarketMakerPairs;

    event UpdateUniswapV2Router(address indexed newAddress);

    event ExcludeFromFees(address indexed account, bool isExcluded);
    event ExcludeMultipleAccountsFromFees(address[] accounts, bool isExcluded);

    event SetAutomatedMarketMakerPair(address indexed pair, bool indexed value);

    event SendFeeToBuyBackContract(address indexed received, uint256 amount);

    constructor(string memory name_, string memory symbol_, uint256 totalSupply_)
ERC20(name_, symbol_) {
        setMinTokensBeforeSwap(500_000 ether);

        excludeFromFees(owner(), true);
        excludeFromFees(address(this), true);

        _mint(owner(), totalSupply_ * (10**18));
    }
```

```
    receive() external payable {}
    //SlowMist// The onwer role can change the minTokensBeforeSwap through the setMinTokensBeforeSwap function
    function setMinTokensBeforeSwap(uint256 amount) public onlyOwner {
        minTokensBeforeSwap = amount;
    }
    //SlowMist// The owner role can update the uniswapV2Router through the updateUniswapV2Router function
    function updateUniswapV2Router(address newAddress) public onlyOwner {
        uniswapV2Router = IUniswapV2Router02(newAddress);
        address _uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory())
        .createPair(address(this), uniswapV2Router.WKLAY());

        uniswapV2Pair = _uniswapV2Pair;
        _setAutomatedMarketMakerPair(_uniswapV2Pair, true);

        emit UpdateUniswapV2Router(newAddress);
    }
    //SlowMist// The owner role can add or remove users from the isExcludedFromFees list, in this list, users are not charged a fee for transferring transactions
    function excludeFromFees(address account, bool excluded) public onlyOwner {
        isExcludedFromFees[account] = excluded;

        emit ExcludeFromFees(account, excluded);
    }

    function excludeMultipleAccountsFromFees(
        address[] calldata accounts,
        bool excluded
    ) public onlyOwner {
        for (uint256 i = 0; i < accounts.length; i++) {
            isExcludedFromFees[accounts[i]] = excluded;
        }

        emit ExcludeMultipleAccountsFromFees(accounts, excluded);
    }
    //SlowMist// The owner role can change the buyBackContract contract through the connectBuyBackContract and the changeBuyBackContract functions
    function connectBuyBackContract(address _buyBackContract) external onlyOwner {
        require(!buyBackContractConnected, "BuyBack contract already connected");

        changeBuyBackContract(_buyBackContract);
        setBuyBackFee(1);
```

```
        buyBackContractConnected = true;
    }

    function changeBuyBackContract(address _buyBackContract) public onlyOwner {
        require(_buyBackContract != address(0), "Wrong address");

        buyBackContract = _buyBackContract;
    }
    //SlowMist// The owner role can set the buyBackFee through the setBuyBackFee
function but the buyBackFee can not exceed 15
    function setBuyBackFee(uint256 value) public onlyOwner {
        buyBackFee = value;

        require(value <= 15, "Cannot exceed max fee limit of 15%");
    }

    function setAutomatedMarketMakerPair(address pair, bool value) public onlyOwner{
        require(pair != uniswapV2Pair, "ERC20DividendToken: The PancakeSwap pair
cannot be removed from automatedMarketMakerPairs");

        _setAutomatedMarketMakerPair(pair, value);
    }

    function _setAutomatedMarketMakerPair(address pair, bool value) private {
        require(automatedMarketMakerPairs[pair] != value, "ERC20DividendToken:
Automated market maker pair is already set to that value");

        automatedMarketMakerPairs[pair] = value;

        emit SetAutomatedMarketMakerPair(pair, value);
    }
    //SlowMist// Suspending all transactions upon major abnormalities is a
recommended approach
    function pause() public onlyOwner {
        _pause();
    }

    function unpause() public onlyOwner {
        _unpause();
    }

    function _burn(address account, uint256 amount) internal whenNotPaused override {
        super._burn(account, amount);
    }
```

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal whenNotPaused override {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(balanceOf(from) >= amount, "ERC20: transfer amount exceeds balance");

    if (amount == 0) {
        super._transfer(from, to, 0);
        return;
    }

    uint256 contractTokenBalance = balanceOf(address(this));

    bool canSwap = contractTokenBalance >= minTokensBeforeSwap;

    if (
        canSwap &&
        !swapping &&
        !automatedMarketMakerPairs[from] &&
        from != owner() &&
        to != owner() &&
        buyBackContractConnected
    ) {
        swapping = true;

        swapAndSendFee(buyBackContract, contractTokenBalance);

        swapping = false;
    }

    bool takeFee = !swapping && buyBackFee > 0;

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

    if (takeFee) {
        uint256 fees = amount.mul(buyBackFee).div(100);
```

```
            if (fees > 0) {
                amount = amount.sub(fees);
                super._transfer(from, address(this), fees);
            }
        }

        super._transfer(from, to, amount);
    }
    //SlowMist// The swapTokensForEth function does not check the slippage when swapping
    function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WKLAY();

        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // make the swap
        uniswapV2Router.swapExactTokensForKLAYSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }

    function swapAndSendFee(address recipient, uint256 tokens) private {
        swapTokensForEth(tokens);

        uint256 dividends = address(this).balance;
        (bool success,) = payable(recipient).call{value: dividends}("");

        if(success) {
            emit SendFeeToBuyBackContract(recipient, dividends);
        }
    }
}

contract CENT is ERC20DividendToken {

    uint256 private _tokenSupply = 5_000_000_000;
```

```
    /**
     * @dev Choose proper router address according to your network:
     * Ethereum mainnet: 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D (Uniswap)
     * KLAYTN mainnet: 0xEf71750C100f7918d6Ded239Ff1CF09E81dEA92D (ClaimSwap)
     */

    constructor () ERC20DividendToken("Center Coin", "CENT", _tokenSupply) {
        // Fees to be set in parent constructor
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist