Smart Contract Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the Lily (LIY) on 2024.07.02. The following are the details and results of this smart contract security audit:

Token Name :

Lily (LIY)

The contract address :

https://polygonscan.com/address/0x57AF7c623766499dE5d0B0C5b74671330CB53F31

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 Audit	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
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
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002407040001

Audit Date : 2024.07.02 - 2024.07.04

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section and dark coin function. The total amount of contract tokens can not be changed, users can burn their tokens through the burn function. The contract does not have the Overflow and the Race Conditions issue.

During the audit,we found the following information:

1. The admin role can freeze the target address token through the setReserve function.

The source code:

```
/**
 *Submitted for verification at polygonscan.com on 2024-06-28
*/

// File: @openzeppelin/contracts/GSN/Context.sol
//SlowMist// The contract does not have the Overflow and the Race Conditions issue.
pragma solidity ^0.5.0;

/*
 * @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 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view 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/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
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

pragma solidity ^0.5.0;

/**
 * @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.
 */
//SlowMist// SafeMath security module is used, which is a recommend approach
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.
     *
     * _Available since v2.4.0._
     */
```

```
    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.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        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.
```

```
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

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

pragma solidity ^0.5.0;




/**
 * @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 {ERC20Mintable}.
 *
 * 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 guidelines: functions revert instead
 * of 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 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;
```

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

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view 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 returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        //SlowMist// The return value conforms to the BEP20 specification
        return true;
    }

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

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        //SlowMist// The return value conforms to the BEP20 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
returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount,
"ERC20: transfer amount exceeds allowance"));
        //SlowMist// The return value conforms to the BEP20 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 returns
(bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()]
[spender].add(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
returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()]
[spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is 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 {
        //SlowMist// This check is quite good in avoiding losing tokens caused by user
mistakes
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount
exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(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
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal {
```

```
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(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 {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount
exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is 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 {
        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 Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount,
"ERC20: burn amount exceeds allowance"));
    }
}

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

pragma solidity ^0.5.0;

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
```

```
}

// File: @openzeppelin/contracts/access/roles/MinterRole.sol

pragma solidity ^0.5.0;



contract MinterRole is Context {
    using Roles for Roles.Role;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    Roles.Role private _minters;

    constructor () internal {
        _addMinter(_msgSender());
    }

    modifier onlyMinter() {
        require(isMinter(_msgSender()), "MinterRole: caller does not have the Minter
role");
        _;
    }

    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }

    function addMinter(address account) public onlyMinter {
        _addMinter(account);
    }

    function renounceMinter() public {
        _removeMinter(_msgSender());
    }

    function _addMinter(address account) internal {
        _minters.add(account);
        emit MinterAdded(account);
    }

    function _removeMinter(address account) internal {
        _minters.remove(account);
        emit MinterRemoved(account);
    }
}
```

```
// File: @openzeppelin/contracts/token/ERC20/ERC20Mintable.sol

pragma solidity ^0.5.0;



/**
 * @dev Extension of {ERC20} that adds a set of accounts with the {MinterRole},
 * which have permission to mint (create) new tokens as they see fit.
 *
 * At construction, the deployer of the contract is the only minter.
 */
contract ERC20Mintable is ERC20, MinterRole {
    /**
     * @dev See {ERC20-_mint}.
     *
     * Requirements:
     *
     * - the caller must have the {MinterRole}.
     */
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Capped.sol

pragma solidity ^0.5.0;



/**
 * @dev Extension of {ERC20Mintable} that adds a cap to the supply of tokens.
 */
contract ERC20Capped is ERC20Mintable {
    uint256 private _cap;

    /**
     * @dev Sets the value of the `cap`. This value is immutable, it can only be
     * set once during construction.
     */
    constructor (uint256 cap) public {
        require(cap > 0, "ERC20Capped: cap is 0");
        _cap = cap;
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
```

```
    function cap() public view returns (uint256) {
        return _cap;
    }

    /**
     * @dev See {ERC20Mintable-mint}.
     *
     * Requirements:
     *
     * - `value` must not cause the total supply to go over the cap.
     */
    //SlowMist// The contract's owner can mint tokens arbitrarily but the mint amount of tokens has an upper limit cap
    function _mint(address account, uint256 value) internal {
        require(totalSupply().add(value) <= _cap, "ERC20Capped: cap exceeded");
        super._mint(account, value);
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Detailed.sol

pragma solidity ^0.5.0;


/**
 * @dev Optional functions from the ERC20 standard.
 */
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
     * these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

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

```
    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view 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.
     *
     * 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 returns (uint8) {
        return _decimals;
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Burnable.sol

pragma solidity ^0.5.0;



/**
 * @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).
 */
contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev See {ERC20-_burnFrom}.
```

```
     */
    //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 {
        _burnFrom(account, amount);
    }
}

// File: contracts/LiyCoin.sol

pragma solidity >=0.5.0<0.6.0;




/**
 * @title coin contract
 */
contract LiyCoin is ERC20Capped, ERC20Detailed, ERC20Burnable {

    // Address of coin vault
    // The vault will have all coin issued.
    address internal vault;

    // Address of  owner
    // The owner can change admin and vault address.
    address internal owner;

    // Address of coin admin
    // The admin can change reserve. The reserve is the amount of token
    // assigned to some address but not permitted to use.
    address internal admin;

    event OwnerChanged(address indexed previousOwner, address indexed newOwner);
    event VaultChanged(address indexed previousVault, address indexed newVault);
    event AdminChanged(address indexed previousAdmin, address indexed newAdmin);
    event ReserveChanged(address indexed _address, uint amount);

    /**
     * @dev reserved number of tokens per each address
     *
     * To limit token transaction for some period by the admin,
     * each address' balance cannot become lower than this amount
     *
     */
    mapping(address => uint) public reserves;

    /**
       * @dev modifier to limit access to the owner only
```

```
       */
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    /**
       * @dev limit access to the vault only
       */
    modifier onlyVault() {
        require(msg.sender == vault);
        _;
    }

    /**
       * @dev limit access to the admin only
       */
    modifier onlyAdmin() {
        require(msg.sender == admin);
        _;
    }

    /**
       * @dev limit access to owner or vault
       */
    modifier onlyOwnerOrVault() {
        require(msg.sender == owner || msg.sender == vault);
        _;
    }

    /**
     * @dev initialize ERC20
     *
     * all token will deposit into the vault
     * later, the vault, owner will be multi sign contract to protect privileged
operations
     *
     * @param _symbol token symbol
     * @param _name   token name
     * @param _total          uint256
     * @param _decimals     uint8
     * @param _owner  owner address
     * @param _admin  admin address
     * @param _vault  vault address
     * @param _cap          uint256
     *
     */
    //SlowMist// require(_owner != address(0));
    //SlowMist// require(_admin != address(0));
```

```
    //SlowMist// require(_vault != address(0));
    //SlowMist// It is recommended to add the above line check to avoid losing control of the contract caused by user mistakes
    //SlowMist// It is recommended to use msg.sender instead of _owner to initialize the owner address because the initial minter will be msg.sender in MinterRole contract constructor function
    constructor (string memory _symbol, string memory _name, uint256 _total, uint8 _decimals, address _owner,
                 address _admin, address _vault, uint256 _cap)
        ERC20Detailed(_name, _symbol, _decimals) ERC20Capped(_cap * (10 ** uint(_decimals)))
    public {
        require(bytes(_symbol).length > 0);
        require(bytes(_name).length > 0);

        owner = _owner;
        admin = _admin;
        vault = _vault;

        // mint coins to the vault
        _mint(vault, _total * (10 ** uint(decimals())));
    }

    /**
     * @dev change the amount of reserved token
     *
     * @param _address the target address whose token will be frozen for future use
     * @param _reserve  the amount of reserved token
     *
     */
    //SlowMist// The contract's admin can freeze the target address token
    function setReserve(address _address, uint _reserve) public onlyAdmin {
        require(_reserve <= totalSupply());
        require(_address != address(0));

        reserves[_address] = _reserve;
        emit ReserveChanged(_address, _reserve);
    }

    /**
     * @dev transfer token from sender to other
     * the result balance should be greater than or equal to the reserved token amount
     */
    function transfer(address _to, uint256 _value) public returns (bool) {
        // check the reserve
        require(balanceOf(msg.sender).sub(_value) >= reserveOf(msg.sender));
        return super.transfer(_to, _value);
    }
```

```
    /**
     * @dev change vault address
     *
     * @param _newVault new vault address
     */
    //SlowMist// The contract's owner can change the vault address
    function setVault(address _newVault) public onlyOwner {
        require(_newVault != address(0));
        require(_newVault != vault);

        address _oldVault = vault;

        // change vault address
        vault = _newVault;
        emit VaultChanged(_oldVault, _newVault);
    }

    /**
     * @dev change owner address
     * @param _newOwner new owner address
     */
    //SlowMist// The contract's vault can change the owner address
    function setOwner(address _newOwner) public onlyVault {
        require(_newOwner != address(0));
        require(_newOwner != owner);

        _addMinter(_newOwner);
        _removeMinter(owner);

        emit OwnerChanged(owner, _newOwner);
        owner = _newOwner;
    }

    /**
     * @dev change admin address
     * @param _newAdmin new admin address
     */
    //SlowMist// The contract's owner or vault can change the admin address
    function setAdmin(address _newAdmin) public onlyOwnerOrVault {
        require(_newAdmin != address(0));
        require(_newAdmin != admin);

        emit AdminChanged(admin, _newAdmin);
        admin = _newAdmin;
    }

    /**
     * @dev Transfer tokens from one address to another
     *
```

```
     * The _from's balance should be larger than the reserved amount(reserves[_from])
plus _value.
     *
     * NOTE: no one can tranfer from vault
     *
     */
    //SlowMist// Only the contract's vault can transfer vault tokens to other
addresses
    function transferFrom(address _from, address _to, uint256 _value) public returns
(bool) {
        require(_from != vault);
        require(_value <= balanceOf(_from).sub(reserves[_from]));
        return super.transferFrom(_from, _to, _value);
    }

    function getOwner() public view returns (address) {
        return owner;
    }

    function getVault() public view returns (address) {
        return vault;
    }

    function getAdmin() public view returns (address) {
        return admin;
    }

    function getOneCoin() public view returns (uint) {
        return (10 ** uint(decimals()));
    }

    /**
     * @dev get the amount of reserved token
     */
    function reserveOf(address _address) public view returns (uint _reserve) {
        return reserves[_address];
    }

    /**
     * @dev get the amount reserved token of the sender
     */
    function reserve() public view returns (uint _reserve) {
        return reserves[msg.sender];
    }
}
```

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