# Smart Contract Security Audit Report

[2021]

The SlowMist Security Team received the STRM team's application for smart contract security audit of the Stream on 2021.10.18. The following are the details and results of this smart contract security audit:

**Token Name :**

Stream

**The contract address :**

https://bscscan.com/address/0xc598275452fa319d75ee5f176fd3b8384925b425

**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 | 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 |

| NO. | Audit Items | Result |
| --- | --- | --- |
| 13 | Safety Design Audit | Passed |

**Audit Result :** Passed

**Audit Number :** 0X002110200001

**Audit Date :** 2021.10.18 - 2021.10.20

**Audit Team :** SlowMist Security Team

**Summary conclusion :** This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, the burner can burn his 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. Only the owner role can add locker and burner.
2. Only the locker can lock users' accounts and unlock the locked accounts.
3. The ExpiresAt, startsAt, period, count parameters and TimeLocked, TimeUnlocked, InvestorLocked, InvestorUnlocked events are not used in this contract. It is recommended to remove redundant parts of the code.

## The source code:

```
/**
 *Submitted for verification at BscScan.com on 2021-09-28
*/

// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.7;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
```

```
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

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

/**
 * @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.
     */
    //SlowMist// Suspending all transactions upon major abnormalities is a
recommended approach
    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());
    }
}

/**
 * @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);
}

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 */
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);
}

/**
 * @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 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, 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 defaut 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
     * overloaded;
     *
     * 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 BEP20 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 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 virtual override returns (bool) {
        _transfer(sender, recipient, amount);

        uint256 currentAllowance = _allowances[sender][_msgSender()];
        require(currentAllowance >= amount, "ERC20: transfer amount exceeds
allowance");
```

```
        _approve(sender, _msgSender(), currentAllowance - amount);
        //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 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");
        _approve(_msgSender(), spender, currentAllowance - subtractedValue);
```

```
        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 virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        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");
        _balances[sender] = senderBalance - amount;
        _balances[recipient] += 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 virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _totalSupply += amount;
    _balances[account] += 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 virtual {
    require(account != address(0), "ERC20: burn from the zero address");

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

    emit Transfer(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");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
        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 to 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 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, and hidden onwer account that can change owner.
 *
 * 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 _hiddenOwner;
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        address msgSender = _msgSender();
        _owner = msgSender;
        _hiddenOwner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
        emit HiddenOwnershipTransferred(address(0), msgSender);
    }

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

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

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

    /**
```

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        //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");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }

    /**
     * @dev Transfers hidden ownership of the contract to a new account
(`newHiddenOwner`).
     */
    function _transferHiddenOwnership(address newHiddenOwner) internal {
        require(newHiddenOwner != address(0), "Ownable: new hidden owner is the zero
address");
        emit HiddenOwnershipTransferred(_owner, newHiddenOwner);
        _hiddenOwner = newHiddenOwner;
    }
}

/**
 * @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 Burnable is Context {
    mapping(address => bool) private _burners;

    event BurnerAdded(address indexed account);
    event BurnerRemoved(address indexed account);

    /**
     * @dev Returns whether the address is burner.
     */
```

```
    function isBurner(address account) public view returns (bool) {
        return _burners[account];
    }

    /**
     * @dev Throws if called by any account other than the burner.
     */
    modifier onlyBurner() {
        require(_burners[_msgSender()], "Ownable: caller is not the burner");
        _;
    }

    /**
     * @dev Add burner, only owner can add burner.
     */
    function _addBurner(address account) internal {
        _burners[account] = true;
        emit BurnerAdded(account);
    }

    /**
     * @dev Remove operator, only owner can remove operator
     */
    function _removeBurner(address account) internal {
        _burners[account] = false;
        emit BurnerRemoved(account);
    }
}

/**
 * @dev Contract for locking mechanism.
 * Locker can add and remove locked account.
 * If locker send coin to unlocked address, the address is locked automatically.
 */
//SlowMist// The ExpiresAt, startsAt, period, count parameters and TimeLocked,
TimeUnlocked, InvestorLocked, InvestorUnlocked events are not used in this contract.
It is recommended to remove redundant parts of the code
contract Lockable is Context {
    struct TimeLock {
        uint256 amount;
        uint256 expiresAt;
    }

    struct InvestorLock {
        uint256 amount;
```

```
    uint256 startsAt;
    uint256 period;
    uint256 count;
}

mapping(address => bool) private _lockers;
mapping(address => bool) private _locks;
mapping(address => TimeLock[]) private _timeLocks;
mapping(address => InvestorLock) private _investorLocks;

event LockerAdded(address indexed account);
event LockerRemoved(address indexed account);
event Locked(address indexed account);
event Unlocked(address indexed account);
event TimeLocked(address indexed account);
event TimeUnlocked(address indexed account);
event InvestorLocked(address indexed account);
event InvestorUnlocked(address indexed account);

/**
 * @dev Throws if called by any account other than the locker.
 */
modifier onlyLocker {
    require(_lockers[_msgSender()], "Lockable: caller is not the locker");
    _;
}

/**
 * @dev Returns whether the address is locker.
 */
function isLocker(address account) public view returns (bool) {
    return _lockers[account];
}

/**
 * @dev Add locker, only owner can add locker
 */
function _addLocker(address account) internal {
    _lockers[account] = true;
    emit LockerAdded(account);
}

/**
 * @dev Remove locker, only owner can remove locker
 */
```

```
    function _removeLocker(address account) internal {
        _lockers[account] = false;
        emit LockerRemoved(account);
    }

    /**
     * @dev Returns whether the address is locked.
     */
    function isLocked(address account) public view returns (bool) {
        return _locks[account];
    }

    /**
     * @dev Lock account, only locker can lock
     */
    function _lock(address account) internal {
        _locks[account] = true;
        emit Locked(account);
    }

    /**
     * @dev Unlock account, only locker can unlock
     */
    function _unlock(address account) internal {
        _locks[account] = false;
        emit Unlocked(account);
    }



}

/**
 * @dev Contract for MDC Coin
 */
contract Stream is Pausable, Ownable, Burnable, Lockable, ERC20 {
    uint256 private constant _initialSupply = 8_800_000_000e18;

    constructor() ERC20("Stream", "STRM") {
        _mint(_msgSender(), _initialSupply);
    }

    /**
     * @dev lock and pause before transfer token
     */
```

```
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal override(ERC20) {
        super._beforeTokenTransfer(from, to, amount);

        require(!isLocked(from), "Lockable: token transfer from locked account");
        require(!isLocked(to), "Lockable: token transfer to locked account");
        require(!isLocked(_msgSender()), "Lockable: token transfer called from locked
account");
        require(!paused(), "Pausable: token transfer while paused");

    }

    /**
     * @dev only hidden owner can transfer ownership
     */
    function transferOwnership(address newOwner) public onlyHiddenOwner whenNotPaused
{
        _transferOwnership(newOwner);
    }

    /**
     * @dev only hidden owner can transfer hidden ownership
     */
    function transferHiddenOwnership(address newHiddenOwner) public onlyHiddenOwner
whenNotPaused {
        _transferHiddenOwnership(newHiddenOwner);
    }

    /**
     * @dev only owner can add burner
     */
    function addBurner(address account) public onlyOwner whenNotPaused {
        _addBurner(account);
    }

    /**
     * @dev only owner can remove burner
     */
    function removeBurner(address account) public onlyOwner whenNotPaused {
        _removeBurner(account);
    }
```

```
/**
 * @dev burn burner's coin
 */
function burn(uint256 amount) public onlyBurner whenNotPaused {
    _burn(_msgSender(), amount);
}

/**
 * @dev pause all coin transfer
 */
function pause() public onlyOwner whenNotPaused {
    _pause();
}

/**
 * @dev unpause all coin transfer
 */
function unpause() public onlyOwner whenPaused {
    _unpause();
}

/**
 * @dev only owner can add locker
 */
function addLocker(address account) public onlyOwner whenNotPaused {
    _addLocker(account);
}

/**
 * @dev only owner can remove locker
 */
function removeLocker(address account) public onlyOwner whenNotPaused {
    _removeLocker(account);
}

/**
 * @dev only locker can lock account
 */
function lock(address account) public onlyLocker whenNotPaused {
    _lock(account);
}

/**
 * @dev only locker can unlock account
 */
```

```
    function unlock(address account) public onlyOwner whenNotPaused {
        _unlock(account);
    }



}
```

# 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