Smart Contract Security Audit Report

The SlowMist Security Team received the OBSR team's application for smart contract security audit of the OBSR Token on January 8, 2020. The following are the details and results of this smart contract security audit:

Token name :

OBSR

The Contract address :

0x87def9265b40ba7f867f73d4af519cd9f074bed6

Link address :

https://etherscan.io/address/0x87def9265b40ba7f867f73d4af519cd9f074bed6

The audit items and results :

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

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Some Risk
8	“False Deposit” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002001150001

Audit Date : January 15, 2020

Audit Team : SlowMist Security Team

(**Statement :** SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that does not contain the tokenVault section. OpenZeppelin' s SafeMath security Module is used, which is a recommend approach. Following problems was found during the audit:

1. In unlock function, the delete operation just clear data of its position but does not change the length of the array, and as more and more people lock their tokens, the reasons array will become larger and larger, and results in large gas cost when unlock tokens and the max recursive depth in EVM is 1024

2. In unlock function, the function does not delete the corresponding address in lockedAddrs, results in duplicate address in lockedAddrs when lock tokens

3. In transferWithLock function, the function does not restrict the _time parameter must larger than 0. results in immediately unlock tokens when transfer tokens to others, and does not restrict _to can not be msg.sender itself as well, which results in duplicate address in lockedAddrs.

The source code:

SafeMath.sol:

```
pragma solidity 0.5.2;

//SlowMist// OpenZeppelin’ s SafeMath security Module is used, which is a recommend

approach

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
```

```
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        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-solidity/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) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, "SafeMath: division by zero");
        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) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}
```

ERC1132.sol:

```
pragma solidity 0.5.2;


/**
 * @title ERC1132 interface
 * @dev see https://github.com/ethereum/EIPs/issues/1132
 */
```

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
contract ERC1132 {
    /**
     * @dev Reasons why a user's tokens have been locked
     */
    mapping(address => bytes32[]) public lockReason;

    /**
     * @dev locked token structure
     */
    struct lockToken {
        uint256 amount;
        uint256 validity;
        bool claimed;
    }

    /**
     * @dev Holds number & validity of tokens locked for a given reason for
     *        a specified address
     */
    mapping(address => mapping(bytes32 => lockToken)) public locked;

    /**
     * @dev Records data of all the tokens Locked
     */
    event Locked(
        address indexed _of,
        bytes32 indexed _reason,
        uint256 _amount,
        uint256 _validity
    );

    /**
     * @dev Records data of all the tokens unlocked
     */
    event Unlocked(
        address indexed _of,
        bytes32 indexed _reason,
        uint256 _amount
    );


function getListLockedAddrs()
```

```
        public view returns( address    [] memory);

        function getListResons()
        public view returns(bytes32 [] memory);

        function getAddrByReason(bytes32 my_reason)
        public view returns(address);


    /**
     * @dev Locks a specified amount of tokens against an address,
     *          for a specified reason and time
     * @param _reason The reason to lock tokens
     * @param _amount Number of tokens to be locked
     * @param _time Lock time in seconds
     */
    function lock(bytes32 _reason, uint256 _amount, uint256 _time)
        public returns (bool);

    /**
     * @dev Returns tokens locked for a specified address for a
     *          specified reason
     *
     * @param _of The address whose tokens are locked
     * @param _reason The reason to query the lock tokens for
     */
    function tokensLocked(address _of, bytes32 _reason)
        public view returns (uint256 amount);

    /**
     * @dev Returns tokens locked for a specified address for a
     *          specified reason at a specific time
     *
     * @param _of The address whose tokens are locked
     * @param _reason The reason to query the lock tokens for
     * @param _time The timestamp to query the lock tokens for
     */
    function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
        public view returns (uint256 amount);

    /**
     * @dev Returns total tokens held by an address (locked + transferable)
```

```
    * @param _of The address to query the total balance of
    */
    function totalBalanceOf(address _of)
        public view returns (uint256 amount);

    /**
    * @dev Extends lock for a specified reason and time
    * @param _reason The reason to lock tokens
    * @param _time Lock extension time in seconds
    */
    function extendLock(bytes32 _reason, uint256 _time)
        public returns (bool);

    /**
    * @dev Increase number of tokens locked for a specified reason
    * @param _reason The reason to lock tokens
    * @param _amount Number of tokens to be increased
    */
    function increaseLockAmount(bytes32 _reason, uint256 _amount)
        public returns (bool);

    /**
    * @dev Returns unlockable tokens for a specified address for a specified reason
    * @param _of The address to query the the unlockable token count of
    * @param _reason The reason to query the unlockable tokens for
    */
    function tokensUnlockable(address _of, bytes32 _reason)
        public view returns (uint256 amount);

    /**
    * @dev Unlocks the unlockable tokens of a specified address
    * @param _of Address of user, claiming back unlockable tokens
    */
    function unlock(address _of)
        public returns (uint256 unlockableTokens);

    /**
    * @dev Gets the unlockable tokens of a specified address
    * @param _of The address to query the the unlockable token count of
    */
    function getUnlockableTokens(address _of)
        public view returns (uint256 unlockableTokens);
```

```
}
```

IERC20.sol:

```
pragma solidity 0.5.2;

/**
 * @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.
    *
    * > 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);
}
```

ERC20.sol:

```
pragma solidity 0.5.2;
```

```
import "./IERC20.sol";
import "./SafeMath.sol";

/**
 * @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`.
 *
 * *For a detailed writeup see our guide [How to implement supply
 * mechanisms](https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226).*
 *
 * 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`.
 */
```

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
contract ERC20 is 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(msg.sender, recipient, amount);

    return true; //SlowMist// The return value conforms to the EIP20 specification

}

/**
 * @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 value) public returns (bool) {
    _approve(msg.sender, spender, value);

    return true; //SlowMist// The return value conforms to the EIP20 specification

}
```

```
/**
 * @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 `value`.
 * - 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, msg.sender, _allowances[sender][msg.sender].sub(amount));
    return true; //SlowMist// The return value conforms to the EIP20 specification
}

/**
 * @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(msg.sender, spender, _allowances[msg.sender][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(msg.sender, spender, _allowances[msg.sender][spender].sub(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 {
        require(sender != address(0), "ERC20: transfer from the zero address");

        require(recipient != address(0), "ERC20: transfer to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

        _balances[sender] = _balances[sender].sub(amount);
        _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"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during mint

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destoys `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.
     */
    //SlowMist// redundancy code
    function _burn(address account, uint256 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
     * @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 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");

        require(spender != address(0), "ERC20: approve to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during approve

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

    /**
     * @dev Destoys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See `_burn` and `_approve`.
     */
    //SlowMist// redundancy code
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
    }
}
```

ERC20Detailed.sol:

```
pragma solidity 0.5.2;

import "./IERC20.sol";

/**
 * @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 that 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;
    }
}
```

OBSR.sol:

```
pragma solidity 0.5.2;


import './ERC1132.sol';
import './ERC20.sol';
import './ERC20Detailed.sol';




/**
 * @title SimpleToken
 * @dev Very simple ERC20 Token example, where all tokens are pre-assigned to the creator.
 * Note they can later distribute these tokens as they wish using `transfer` and other
 * `ERC20` functions.
 */
contract OBSR is   ERC1132, ERC20, ERC20Detailed {
    uint8 public constant DECIMALS = 8;
    uint256 public constant INITIAL_SUPPLY = 15000000000000000000;

/**
    * @dev Error messages for require statements
    */


    string internal constant ALREADY_LOCKED = 'Tokens already locked';
    string internal constant NOT_LOCKED = 'No tokens locked';
    string internal constant AMOUNT_ZERO = 'Amount can not be 0';

    address[] lockedAddrs;
    bytes32[] reasons;
```

```
    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */

    constructor () public ERC20Detailed("OBSR", "OBSR", DECIMALS) {
        _mint(msg.sender, INITIAL_SUPPLY);
    }

function getListLockedAddrs()public view returns( address    [] memory)
{
      return lockedAddrs;
}


function getListResons()public view returns(bytes32 [] memory)
{
      return reasons;
}


//returns addrlocked by reason




function getAddrByReason(bytes32 my_reason) public view returns(address)
{
    for (uint256 j = 0; j < lockedAddrs.length; j++)
        {
            uint256 amountLocked =     tokensLockedAtTime(lockedAddrs[j],my_reason,now);

            if(amountLocked!=0)
            {
                return     lockedAddrs[j];
            }
        }
}

function tokensValidityLockedAtTime(address _of, bytes32 _reason, uint256 _time)
        public
```

```
        view
        returns (uint256 validity)
    {
        if (locked[_of][_reason].validity > _time)
            validity = locked[_of][_reason].validity;
    }



/**
     * @dev Locks a specified amount of tokens against an address,
     *      for a specified reason and time
     * @param _reason The reason to lock tokens
     * @param _amount Number of tokens to be locked
     * @param _time Lock time in seconds
     */


    function lock(bytes32 _reason, uint256 _amount, uint256 _time)
        public
        returns (bool)
    {
        uint256 validUntil = now.add(_time); //solhint-disable-line

        // If tokens are already locked, then functions extendLock or
        // increaseLockAmount should be used to make any changes
        require(tokensLocked(msg.sender, _reason) == 0, ALREADY_LOCKED);
        require(_amount != 0, AMOUNT_ZERO);

        if (locked[msg.sender][_reason].amount == 0)
            lockReason[msg.sender].push(_reason);

        transfer(address(this), _amount);

        locked[msg.sender][_reason] = lockToken(_amount, validUntil, false);

        lockedAddrs.push(msg.sender);
```

```
        reasons.push(_reason);

        emit Locked(msg.sender, _reason, _amount, validUntil);
        return true;
    }

    /**
     * @dev Transfers and Locks a specified amount of tokens,
     *      for a specified reason and time
     * @param _to adress to which tokens are to be transfered
     * @param _reason The reason to lock tokens
     * @param _amount Number of tokens to be transfered and locked
     * @param _time Lock time in seconds
     */
    function transferWithLock(address _to, bytes32 _reason, uint256 _amount, uint256 _time)
        public
        returns (bool)
    {
        uint256 validUntil = now.add(_time); //solhint-disable-line

        require(tokensLocked(_to, _reason) == 0, ALREADY_LOCKED);
        require(_amount != 0, AMOUNT_ZERO);

        if (locked[_to][_reason].amount == 0)
            lockReason[_to].push(_reason);

        transfer(address(this), _amount);

        locked[_to][_reason] = lockToken(_amount, validUntil, false);

        emit Locked(_to, _reason, _amount, validUntil);

        lockedAddrs.push(_to);
        reasons.push(_reason);

        return true;
    }

    /**
     * @dev Returns tokens locked for a specified address for a
     *      specified reason
     *
```

```
     * @param _of The address whose tokens are locked
     * @param _reason The reason to query the lock tokens for
     */
    function tokensLocked(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (!locked[_of][_reason].claimed)
            amount = locked[_of][_reason].amount;
    }

    /**
     * @dev Returns tokens locked for a specified address for a
     *      specified reason at a specific time
     *
     * @param _of The address whose tokens are locked
     * @param _reason The reason to query the lock tokens for
     * @param _time The timestamp to query the lock tokens for
     */
    function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
        public
        view
        returns (uint256 amount)
    {
        if (locked[_of][_reason].validity > _time)
            amount = locked[_of][_reason].amount;
    }

    /**
     * @dev Returns total tokens held by an address (locked + transferable)
     * @param _of The address to query the total balance of
     */
    function totalBalanceOf(address _of)
        public
        view
        returns (uint256 amount)
    {
        amount = balanceOf(_of);

        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
```

```
        }
    }

    /**
     * @dev Extends lock for a specified reason and time
     * @param _reason The reason to lock tokens
     * @param _time Lock extension time in seconds
     */
    function extendLock(bytes32 _reason, uint256 _time)
        public
        returns (bool)
    {
        require(tokensLocked(msg.sender, _reason) > 0, NOT_LOCKED);

        locked[msg.sender][_reason].validity = locked[msg.sender][_reason].validity.add(_time);

        emit Locked(msg.sender, _reason, locked[msg.sender][_reason].amount, locked[msg.sender][_reason].validity);
        return true;
    }

    /**
     * @dev Increase number of tokens locked for a specified reason
     * @param _reason The reason to lock tokens
     * @param _amount Number of tokens to be increased
     */
    function increaseLockAmount(bytes32 _reason, uint256 _amount)
        public
        returns (bool)
    {
        require(tokensLocked(msg.sender, _reason) > 0, NOT_LOCKED);
        transfer(address(this), _amount);

        locked[msg.sender][_reason].amount = locked[msg.sender][_reason].amount.add(_amount);

        emit Locked(msg.sender, _reason, locked[msg.sender][_reason].amount, locked[msg.sender][_reason].validity);
        return true;
    }

    /**
     * @dev Returns unlockable tokens for a specified address for a specified reason
     * @param _of The address to query the the unlockable token count of
     * @param _reason The reason to query the unlockable tokens for
```

```
    */
function tokensUnlockable(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
{
        if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-disable-line
            amount = locked[_of][_reason].amount;
}

/**
    * @dev Unlocks the unlockable tokens of a specified address
    * @param _of Address of user, claiming back unlockable tokens
    */
function unlock(address _of)
        public
        returns (uint256 unlockableTokens)
{
        uint256 lockedTokens;

        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
            if (lockedTokens > 0) {
                unlockableTokens = unlockableTokens.add(lockedTokens);
                locked[_of][lockReason[_of][i]].claimed = true;
                emit Unlocked(_of, lockReason[_of][i], lockedTokens);
                        for (uint256 k = 0; k < reasons.length; k++) {
                if(lockReason[_of][i]==reasons[k])
                  delete reasons[k];
                }
            }
        }
        //yudan// 没有删除 lockedAddrs
        if (unlockableTokens > 0)
            this.transfer(_of, unlockableTokens);
}

/**
    * @dev Gets the unlockable tokens of a specified address
    * @param _of The address to query the the unlockable token count of
    */
function getUnlockableTokens(address _of)
```

```
        public
        view
        returns (uint256 unlockableTokens)
    {
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i]));
        }
    }

}



/*

function append(string memory a, string memory b, string memory c) internal pure returns (string memory ) {

    return string(abi.encodePacked(a, b, c));

}


function toString(address x) internal pure returns (string memory) {
        bytes memory b = new bytes(20);
        for (uint i = 0; i < 20; i++)
            b[i] = byte(uint8(uint(x) / (2**(8*(19 - i)))));
        return string(b);
    }


function uint2str(uint _i) internal pure returns (string memory _uintAsString) {
    if (_i == 0) {
        return "0";
    }
    uint j = _i;
    uint len;
    while (j != 0) {
        len++;
        j /= 10;
    }
    bytes memory bstr = new bytes(len);
    uint k = len - 1;
```

```
    while ( _i != 0) {
        bstr[k--] = byte(uint8(48 + _i % 10));
        _i /= 10;
    }
    return string(bstr);
}
*/


/*
   bytes32[]   bts;

function getInfoByReason(bytes32 my_reason) public     returns(bytes32[] memory)
{

        bts =0;
         for (uint256 j = 0; j < lockedAddrs.length; j++)
         {
                uint256 amountLocked =     tokensLockedAtTime(lockedAddrs[j],my_reason,now);

                if(amountLocked!=0)
                {
                        //   string memory abcde = new string(_ba.length + _
                  //   return    append(toString(lockedAddrs[j]), "", uint2str(amountLocked));

                    bts.push(bytes32(uint256(msg.sender)));
                    bts.push(bytes32(amountLocked));

                  return bts;
                  // return "123";
                }


         }



}

*/
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account