Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the Favor on 2022.05.19. The following are the details and results of this smart contract security audit:

Token Name :

Favor

The contract address :

https://scope.klaytn.com/account/0xe79efff8a61567d932be2a8c33057f7b2a8bc43b

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
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002205230001

Audit Date : 2022.05.19 - 2022.05.23

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that contains the tokenVault section. The total amount of contract tokens can be changed, useres can burn their tokens through the burn function.

During the audit, we found the following risk and information:

1. The locker role can lock his own tokens at a specified period of time.
2. The locker role can transfer and lock tokens at a specified period of time.
3. The extendLock function is used to extend the lock for the specified time in seconds. The locker role can call extendLock with a large _time parameter, which may lead to overflow and early unlocking.

The project team has renounced the locker role, and the vulnerable extendLock function cannot be called and exploited. Tokens cannot be locked now.

The source code:

```
// Sources flattened with hardhat v2.9.3 https://hardhat.org

// File contracts/klaytn/introspection/IKIP13.sol
//SlowMist// The contract does not have the Overflow and the Race
pragma solidity ^0.5.0;

/**
 * @dev Interface of the KIP-13 standard, as defined in the
 * [KIP-13](http://kips.klaytn.com/KIPs/kip-13-interface_query_standard).
 *
```

```
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others.
 *
 * For an implementation, see `KIP13`.
 */
interface IKIP13 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * [KIP-13 section](http://kips.klaytn.com/KIPs/kip-13-interface_query_standard#how-interface-identifiers-are-defined)
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}


// File contracts/klaytn/token/KIP7/IKIP7.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the KIP7 standard as defined in the KIP. Does not include
 * the optional functions; to access them see `KIP7Metadata`.
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract IKIP7 is IKIP13 {
    /**
     * @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 Moves `amount` tokens from the caller's account to `recipient`.
    */
```

```
    function safeTransfer(address recipient, uint256 amount, bytes memory data) public;

    /**
    * @dev  Moves `amount` tokens from the caller's account to `recipient`.
    */
    function safeTransfer(address recipient, uint256 amount) public;

    /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
    * `amount` is then deducted from the caller's allowance.
    */
    function safeTransferFrom(address sender, address recipient, uint256 amount, bytes memory
data) public;

    /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
    * `amount` is then deducted from the caller's allowance.
    */
    function safeTransferFrom(address sender, address recipient, uint256 amount) public;

    /**
     * @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 contracts/klaytn/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 recommended 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 contracts/klaytn/introspection/KIP13.sol

pragma solidity ^0.5.0;

/**
 * @dev Implementation of the `IKIP13` interface.
 *
 * Contracts may inherit from this and call `_registerInterface` to declare
 * their support of an interface.
 */
contract KIP13 is IKIP13 {
    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_KIP13 = 0x01ffc9a7;

    /**
```

```
     * @dev Mapping of interface ids to whether or not it's supported.
     */
    mapping(bytes4 => bool) private _supportedInterfaces;

    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for KIP13 itself here
        _registerInterface(_INTERFACE_ID_KIP13);
    }

    /**
     * @dev See `IKIP13.supportsInterface`.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by
     * `interfaceId`. Support of the actual KIP13 interface is automatic and
     * registering its interface id is not required.
     *
     * See `IKIP13.supportsInterface`.
     *
     * Requirements:
     *
     * - `interfaceId` cannot be the KIP13 invalid interface (`0xffffffff`).
     */
    function _registerInterface(bytes4 interfaceId) internal {
        require(interfaceId != 0xffffffff, "KIP13: invalid interface id");
        _supportedInterfaces[interfaceId] = true;
    }
}


// File contracts/klaytn/utils/Address.sol

pragma solidity ^0.5.0;

/**
 * @dev Collection of functions related to the address type,
 */
```

```
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * > It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
}


// File contracts/klaytn/token/KIP7/IKIP7Receiver.sol

pragma solidity ^0.5.0;

/**
 * @title KIP-7 Fungible Token Standard, optional wallet interface
 * @dev Note: the KIP-13 identifier for this interface is 0x9d188c22.
 * see http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract IKIP7Receiver {
    /**
     * @notice Handle the receipt of KIP-7 token
     * @dev The KIP-7 smart contract calls this function on the recipient
     *  after a `safeTransfer`. This function MAY throw to revert and reject the
     *  transfer. Return of other than the magic value MUST result in the
     *  transaction being reverted.
     *  Note: the contract address is always the message sender.
     * @param _operator The address which called `safeTransferFrom` function
     * @param _from The address which previously owned the token
     * @param _amount The token amount which is being transferred.
```

```
     * @param _data Additional data with no specified format
     * @return `bytes4(keccak256("onKIP7Received(address,address,uint256,bytes)"))`
     *  unless throwing
     */
    function onKIP7Received(address _operator, address _from, uint256 _amount, bytes memory
_data) public returns (bytes4);
}


// File contracts/klaytn/token/KIP7/KIP7.sol

pragma solidity ^0.5.0;




/**
 * @dev Implementation of the `IKIP7` 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 `KIP7Mintable`.
 *
 * 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 KIP7 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 KIP may not emit
 * these events, as it isn't required by the specification.
 *
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract KIP7 is KIP13, IKIP7 {
    using SafeMath for uint256;
    using Address for address;

    // Equals to `bytes4(keccak256("onKIP7Received(address,address,uint256,bytes)"))`
    // which can be also obtained as `IKIP7Receiver(0).onKIP7Received.selector`
    bytes4 private constant _KIP7_RECEIVED = 0x9d188c22;

    mapping (address => uint256) private _balances;
```

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

    uint256 private _totalSupply;

    /*
     *     bytes4(keccak256('totalSupply()')) == 0x18160ddd
     *     bytes4(keccak256('balanceOf(address)')) == 0x70a08231
     *     bytes4(keccak256('transfer(address,uint256)')) == 0xa9059cbb
     *     bytes4(keccak256('allowance(address,address)')) == 0xdd62ed3e
     *     bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
     *     bytes4(keccak256('transferFrom(address,address,uint256)')) == 0x23b872dd
     *     bytes4(keccak256("safeTransfer(address,uint256)")) == 0x423f6cef
     *     bytes4(keccak256("safeTransfer(address,uint256,bytes)")) == 0xeb795549
     *     bytes4(keccak256("safeTransferFrom(address,address,uint256)")) == 0x42842e0e
     *     bytes4(keccak256("safeTransferFrom(address,address,uint256,bytes)")) == 0xb88d4fde
     *
     *     => 0x18160ddd ^ 0x70a08231 ^ 0xa9059cbb ^ 0xdd62ed3e ^ 0x095ea7b3 ^ 0x23b872dd ^
0x423f6cef ^ 0xeb795549 ^ 0x42842e0e ^ 0xb88d4fde == 0x65787371
     */
    bytes4 private constant _INTERFACE_ID_KIP7 = 0x65787371;

    constructor () public {
        // register the supported interfaces to conform to KIP7 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7);
    }

    /**
     * @dev See `IKIP7.totalSupply`.
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See `IKIP7.balanceOf`.
     */
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See `IKIP7.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);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    /**
     * @dev See `IKIP7.allowance`.
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See `IKIP7.approve`.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    /**
     * @dev See `IKIP7.transferFrom`.
     *
     * Emits an `Approval` event indicating the updated allowance. This is not
     * required by the KIP. See the note at the beginning of `KIP7`;
     *
     * 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));
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    /**
    * @dev  Moves `amount` tokens from the caller's account to `recipient`.
    */
    function safeTransfer(address recipient, uint256 amount) public {
        safeTransfer(recipient, amount, "");
    }

    /**
    * @dev Moves `amount` tokens from the caller's account to `recipient`.
    */
    function safeTransfer(address recipient, uint256 amount, bytes memory data) public {
        transfer(recipient, amount);
        require(_checkOnKIP7Received(msg.sender, recipient, amount, data), "KIP7: transfer to
non KIP7Receiver implementer");
    }

    /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
    * `amount` is then deducted from the caller's allowance.
    */
    function safeTransferFrom(address sender, address recipient, uint256 amount) public {
        safeTransferFrom(sender, recipient, amount, "");
    }

    /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
    * `amount` is then deducted from the caller's allowance.
    */
    function safeTransferFrom(address sender, address recipient, uint256 amount, bytes memory
data) public {
        transferFrom(sender, recipient, amount);
        require(_checkOnKIP7Received(sender, recipient, amount, data), "KIP7: transfer to non
KIP7Receiver implementer");
    }

    /**
     * @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), "KIP7: 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), "KIP7: transfer to the zero address");

        _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), "KIP7: 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 value) internal {
        require(account != address(0), "KIP7: 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 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), "KIP7: approve from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading to approve
errors
        require(spender != address(0), "KIP7: approve to the zero address");

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

    /**
     * @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, msg.sender, _allowances[account][msg.sender].sub(amount));
    }

    /**
     * @dev Internal function to invoke `onKIP7Received` on a target address.
     * The call is not executed if the target address is not a contract.
     */
    function _checkOnKIP7Received(address sender, address recipient, uint256 amount, bytes
memory _data)
        internal returns (bool)
    {
        if (!recipient.isContract()) {
            return true;
        }

        bytes4 retval = IKIP7Receiver(recipient).onKIP7Received(msg.sender, sender, amount,
_data);
        return (retval == _KIP7_RECEIVED);
    }
}


// File contracts/klaytn/token/KIP7/KIP7Burnable.sol

pragma solidity ^0.5.0;


/**
 * @dev Extension of `KIP7` 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).
 *
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract KIP7Burnable is KIP13, KIP7 {
    /*
     *     bytes4(keccak256('burn(uint256)')) == 0x42966c68
     *     bytes4(keccak256('burnFrom(address,uint256)')) == 0x79cc6790
     *
     *     => 0x42966c68 ^ 0x79cc6790 == 0x3b5a0bf8
     */
    bytes4 private constant _INTERFACE_ID_KIP7BURNABLE = 0x3b5a0bf8;
```

```
    constructor () public {
        // register the supported interfaces to conform to KIP17 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7BURNABLE);
    }

    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See `KIP7._burn`.
     */
    function burn(uint256 amount) public {
        _burn(msg.sender, amount);
    }

    /**
     * @dev See `KIP7._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/klaytn/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 contracts/klaytn/access/roles/PauserRole.sol

pragma solidity ^0.5.0;

contract PauserRole {
    using Roles for Roles.Role;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(msg.sender);
    }

    modifier onlyPauser() {
        require(isPauser(msg.sender), "PauserRole: caller does not have the Pauser role");
        _;
    }

    function isPauser(address account) public view returns (bool) {
```

```
        return _pausers.has(account);
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function renouncePauser() public {
        _removePauser(msg.sender);
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}


// File contracts/klaytn/lifecycle/Pausable.sol

pragma solidity ^0.5.0;

/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
contract Pausable is PauserRole {
    /**
     * @dev Emitted when the pause is triggered by a pauser (`account`).
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by a pauser (`account`).
```

```
 */
event Unpaused(address account);

bool private _paused;

/**
 * @dev Initializes the contract in unpaused state. Assigns the Pauser role
 * to the deployer.
 */
constructor () internal {
    _paused = false;
}

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

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

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

/**
 * @dev Called by a pauser to pause, triggers stopped state.
 */
function pause() public onlyPauser whenNotPaused {
    _paused = true;
    emit Paused(msg.sender);
}
```

```
    /**
     * @dev Called by a pauser to unpause, returns to normal state.
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}


// File contracts/klaytn/token/KIP7/KIP7Pausable.sol

pragma solidity ^0.5.0;



/**
 * @title Pausable token
 * @dev KIP7 modified with pausable transfers.
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract KIP7Pausable is KIP13, KIP7, Pausable {
    /*
     *     bytes4(keccak256('paused()')) == 0x5c975abb
     *     bytes4(keccak256('pause()')) == 0x8456cb59
     *     bytes4(keccak256('unpause()')) == 0x3f4ba83a
     *     bytes4(keccak256('isPauser(address)')) == 0x46fbf68e
     *     bytes4(keccak256('addPauser(address)')) == 0x82dc1ec4
     *     bytes4(keccak256('renouncePauser()')) == 0x6ef8d66d
     *
     *     => 0x5c975abb ^ 0x8456cb59 ^ 0x3f4ba83a ^ 0x46fbf68e ^ 0x82dc1ec4 ^ 0x6ef8d66d ==
0x4d5507ff
     */
    bytes4 private constant _INTERFACE_ID_KIP7PAUSABLE = 0x4d5507ff;

    constructor () public {
        // register the supported interfaces to conform to KIP17 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7PAUSABLE);
    }

    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }
```

```
    function transferFrom(address from, address to, uint256 value) public whenNotPaused
returns (bool) {
        return super.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) public whenNotPaused returns (bool) {
        return super.approve(spender, value);
    }
}


// File contracts/klaytn/token/KIP7/KIP7Metadata.sol

pragma solidity ^0.5.0;


/**
 * @dev Optional functions from the KIP7 standard.
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract KIP7Metadata is KIP13, IKIP7 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /*
     *     bytes4(keccak256('name()')) == 0x06fdde03
     *     bytes4(keccak256('symbol()')) == 0x95d89b41
     *     bytes4(keccak256('decimals()')) == 0x313ce567
     *
     *     => 0x06fdde03 ^ 0x95d89b41 ^ 0x313ce567 == 0xa219a025
     */
    bytes4 private constant _INTERFACE_ID_KIP7_METADATA = 0xa219a025;

    /**
     * @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;
```

```
        // register the supported interfaces to conform to KIP7 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7_METADATA);
    }

    /**
     * @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
     * `IKIP7.balanceOf` and `IKIP7.transfer`.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}


// File contracts/interfaces/IKIP7Lockable.sol

// SPDX-License-Identifier: NONE
pragma solidity ^0.5.0;

/**
 * @title IKIP7Lockable
```

```
 * @dev KIP7 토큰의 락업 기능 추가를 위한 Lockable 인터페이스
 */
interface IKIP7Lockable {
  /**
   * @dev locked token structure
   */
  struct LockToken {
    uint256 amount;
    uint256 validity;
    bool claimed;
  }

  /**
   * @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);

  /**
   * @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
  ) external returns (bool);

  /**
   * @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 Specific time in seconds
 */
function transferWithLock(
  address _to,
  bytes32 _reason,
  uint256 _amount,
  uint256 _time
) external returns (bool);

/**
 * @dev batch transfer with lock
 * @param _toAddrArr[]
 * @param _reasonArr[]
 * @param _amountArr[]
 * @param _timeArr[]
 */
function batchTransferWithLock(
  address[] calldata _toAddrArr,
  bytes32[] calldata _reasonArr,
  uint256[] calldata _amountArr,
  uint256[] calldata _timeArr
) external returns (bool[] memory resultArr);

/**
 * @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)
  external
  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
) external 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) external 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) external 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)
  external
  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)
  external
  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) external returns (uint256 unlockableTokens);
```

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

  /**
   * @dev Gets the length of lock reason of a specified address
   * @param _of The address to query the length of lock reason
   */
  function getLockReasonLength(address _of)
    external
    view
    returns (uint256 length);
}


// File contracts/access/roles/LockerRole.sol

// SPDX-License-Identifier: NONE
pragma solidity ^0.5.0;

contract LockerRole {
  using Roles for Roles.Role;

  Roles.Role private _lockers;

  event LockerAdded(address indexed account);
  event LockerRemoved(address indexed account);

  modifier onlyLocker() {
    require(
      isLocker(msg.sender),
      'LockerRole: caller does not have the Locker role'
    );
    _;
  }

  constructor() public {
    _addLocker(msg.sender);
```

```
  }

  function addLocker(address account) public onlyLocker {
    _addLocker(account);
  }

  function renounceLocker() public {
    _removeLocker(msg.sender);
  }

  function isLocker(address account) public view returns (bool) {
    return _lockers.has(account);
  }

  function _addLocker(address account) internal {
    _lockers.add(account);
    emit LockerAdded(account);
  }

  function _removeLocker(address account) internal {
    _lockers.remove(account);
    emit LockerRemoved(account);
  }
}


// File contracts/token/KIP7/KIP7Lockable.sol

// SPDX-License-Identifier: NONE
pragma solidity ^0.5.0;




/**
 * @dev 핑거랩스 코인 분배 시 락업 기간 설정을 위한 락업 기능을 제공하는 토큰 컨트랙트
 */
contract KIP7Lockable is KIP13, KIP7, LockerRole, IKIP7Lockable {
  /*
   *     bytes4(keccak256('lock(bytes32,uint256,uint256)')) == 0x2e82aaf2
   *     bytes4(keccak256('transferWithLock(address,bytes32,uint256,uint256)')) == 0x4cb5465f
   *     bytes4(keccak256('tokensLocked(address,bytes32)')) == 0x5ca48d8c
   *     bytes4(keccak256('tokensLockedAtTime(address,bytes32,uint256)')) == 0x179e91f1
   *     bytes4(keccak256('totalBalanceOf(address)')) == 0x4b0ee02a
```

```
   *     bytes4(keccak256('extendLock(bytes23,uint256)')) == 0x6b045400
   *     bytes4(keccak256('increaseLockAmount(bytes23,uint256)')) == 0xe38b4c1a
   *     bytes4(keccak256('tokensUnlockable(address,bytes32)')) == 0x5294d0e8
   *     bytes4(keccak256('unlock(address)')) == 0x2f6c493c
   *     bytes4(keccak256('getUnlockableTokens(address)')) == 0xab4a2eb3
   *
   *     => 0x2e82aaf2 ^ 0x4cb5465f ^ 0x5ca48d8c ^ 0x179e91f1 ^ 0x4b0ee02a ^ 0x6b045400 ^
0xe38b4c1a ^ 0x5294d0e8 ^ 0x2f6c493c ^ 0xab4a2eb3 == 0x3c3ebf87
   */
  bytes4 private constant _INTERFACE_ID_KIP7LOCKABLE = 0x3c3ebf87;

  /**
   * @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';

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

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

  constructor() public {
    // register the supported interfaces to conform to KIP17 via KIP13
    _registerInterface(_INTERFACE_ID_KIP7LOCKABLE);
  }

  /**
   * @dev Gets the length of lock reason of a specified address
   * @param _of The address to query the length of lock reason
   */
  function getLockReasonLength(address _of)
    external
    view
    returns (uint256 length)
  {
    length = lockReason[_of].length;
```

```
  }

  /**
   * @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 Specific lock time in seconds
   */
  //SlowMist// The locker role can lock his own tokens at a specified period of time
  function lock(
    bytes32 _reason,
    uint256 _amount,
    uint256 _time
  ) public onlyLocker returns (bool) {
    require(block.timestamp < _time, 'Lock time must be in the future'); //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, _time, false);

    emit Locked(msg.sender, _reason, _amount, _time);
    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 Specific time in seconds
   */
  //SlowMist// The locker role can transfer and lock tokens at a specified period of time
  function transferWithLock(
```

```
    address _to,
    bytes32 _reason,
    uint256 _amount,
    uint256 _time
  ) public onlyLocker returns (bool) {
    require(block.timestamp < _time, 'Lock time must be in the future'); //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, _time, false);

    emit Locked(_to, _reason, _amount, _time);
    return true;
  }

  /**
   * @dev batch transfer with lock
   * @param _toAddrArr[]
   * @param _reasonArr[]
   * @param _amountArr[]
   * @param _timeArr[]
   */
  function batchTransferWithLock(
    address[] calldata _toAddrArr,
    bytes32[] calldata _reasonArr,
    uint256[] calldata _amountArr,
    uint256[] calldata _timeArr
  ) external onlyLocker returns (bool[] memory) {
    require(_toAddrArr.length > 0, '_toAddrArr must not be empty');
    bool[] memory resultArr = new bool[](_toAddrArr.length);

    uint256 totalLength = resultArr.length;
    require(_reasonArr.length == totalLength, 'reason length does not match');
    require(_amountArr.length == totalLength, 'amount length does not match');
    require(_timeArr.length == totalLength, 'time length does not match');

    for (uint256 i = 0; i < _toAddrArr.length; i++) {
      bool result = transferWithLock(
```

```
        _toAddrArr[i],
        _reasonArr[i],
        _amountArr[i],
        _timeArr[i]
      );

      resultArr[i] = result;
    }

    return resultArr;
  }

  /**
   * @dev Extends lock for a specified reason and time
   * @param _reason The reason to lock tokens
   * @param _time Lock extension time in seconds
   */
  //SlowMist// The locker role can call extendLock with a large _time parameter, which may cause overflow and unlock in advance
  function extendLock(bytes32 _reason, uint256 _time)
    public
    onlyLocker
    returns (bool)
  {
    require(tokensLocked(msg.sender, _reason) > 0, NOT_LOCKED);

    locked[msg.sender][_reason].validity =
      locked[msg.sender][_reason].validity +
      _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
  onlyLocker
  returns (bool)
{
  require(tokensLocked(msg.sender, _reason) > 0, NOT_LOCKED);
  transfer(address(this), _amount);

  locked[msg.sender][_reason].amount =
    locked[msg.sender][_reason].amount +
    _amount;

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

/**
 * @dev Unlocks the unlockable tokens of a specified address
 * @param _of Address of user, claiming back unlockable tokens
 */
function unlock(address _of)
  public
  onlyLocker
  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 + lockedTokens;
      locked[_of][lockReason[_of][i]].claimed = true;
      emit Unlocked(_of, lockReason[_of][i], lockedTokens);
    }
  }

  if (unlockableTokens > 0) this.transfer(_of, unlockableTokens);
}
```

```
/**
 * @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 + tokensLocked(_of, lockReason[_of][i]);
  }
}
```

```
  /**
   * @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 <= block.timestamp && // solhint-disable-line
      !locked[_of][_reason].claimed
    )
      //solhint-disable-line
      amount = locked[_of][_reason].amount;
  }

  /**
   * @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 +
        tokensUnlockable(_of, lockReason[_of][i]);
    }
  }
}


// File contracts/klaytn/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
```

```
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be aplied to your functions to restrict their use to
 * the owner.
 */
contract Ownable {
    address payable private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

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

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
```

```
     * > Note: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address payable newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address payable 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;
    }
}


// File contracts/Favor.sol

// SPDX-License-Identifier: NONE
pragma solidity ^0.5.0;




contract Favor is
  KIP7Burnable,
  KIP7Pausable,
  KIP7Metadata,
  KIP7Lockable,
  Ownable
```

```
{
  uint256 public constant FAVOR_SUPPLY_LIMIT = 3e8 ether; // 3억개

  constructor(
    string memory name,
    string memory symbol,
    uint8 decimal
  ) public KIP7Metadata(name, symbol, decimal) KIP7Lockable() Ownable() {
    _mint(msg.sender, FAVOR_SUPPLY_LIMIT);
  }
}
```

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