Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the FAVOR(FAVR) on 2023.06.08. The following are the details and results of this smart contract security audit:

Token Name :

FAVOR(FAVR)

The contract address :

https://scope.klaytn.com/account/0x37e35406c8d87ae243932bf4c9a2138c2b93c8fa?tabId=contractCode

The audit items and results :

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

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

Audit Result : Passed

Audit Number : 0X002306120001

Audit Date : 2023.06.08 - 2023.06.12

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section and dark coin functions. The total amount of contract tokens remains unchangeable. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
// Sources flattened with hardhat v2.14.1 https://hardhat.org
// SPDX-License-Identifier: MIT
// File contracts/access/IAccessControl.sol
// OpenZeppelin Contracts v4.4.1 (access/IAccessControl.sol)
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;

/**
 * @dev External interface of AccessControl declared to support ERC165 detection.
 */
interface IAccessControl {
    /**
     * @dev Emitted when `newAdminRole` is set as ``role``'s admin role, replacing `previousAdminRole`
     *
     * `DEFAULT_ADMIN_ROLE` is the starting admin for all roles, despite
     * {RoleAdminChanged} not being emitted signaling this.
     *
     * _Available since v3.1._
     */
    event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole, bytes32 indexed newAdminRole);

    /**
     * @dev Emitted when `account` is granted `role`.
     *
     * `sender` is the account that originated the contract call, an admin role
     * bearer except when using {AccessControl-_setupRole}.
     */
    event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender);

    /**
```

```
     * @dev Emitted when `account` is revoked `role`.
     *
     * `sender` is the account that originated the contract call:
     *   - if using `revokeRole`, it is the admin role bearer
     *   - if using `renounceRole`, it is the role bearer (i.e. `account`)
     */
    event RoleRevoked(bytes32 indexed role, address indexed account, address indexed
sender);

    /**
     * @dev Returns `true` if `account` has been granted `role`.
     */
    function hasRole(bytes32 role, address account) external view returns (bool);

    /**
     * @dev Returns the admin role that controls `role`. See {grantRole} and
     * {revokeRole}.
     *
     * To change a role's admin, use {AccessControl-_setRoleAdmin}.
     */
    function getRoleAdmin(bytes32 role) external view returns (bytes32);

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function grantRole(bytes32 role, address account) external;

    /**
     * @dev Revokes `role` from `account`.
     *
     * If `account` had been granted `role`, emits a {RoleRevoked} event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function revokeRole(bytes32 role, address account) external;

    /**
     * @dev Revokes `role` from the calling account.
     *
     * Roles are often managed via {grantRole} and {revokeRole}: this function's
```

```
     * purpose is to provide a mechanism for accounts to lose their privileges
     * if they are compromised (such as when a trusted device is misplaced).
     *
     * If the calling account had been granted `role`, emits a {RoleRevoked}
     * event.
     *
     * Requirements:
     *
     * - the caller must be `account`.
     */
    function renounceRole(bytes32 role, address account) external;
}

// File contracts/utils/Context.sol
// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)
pragma solidity ^0.8.0;
/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}

// File contracts/utils/introspection/IERC165.sol
// OpenZeppelin Contracts v4.4.1 (utils/introspection/IERC165.sol)
pragma solidity ^0.8.0;
/**
 * @dev Interface of the ERC165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[EIP].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
```

```
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP
section]
     * 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/utils/introspection/ERC165.sol
// OpenZeppelin Contracts v4.4.1 (utils/introspection/ERC165.sol)
pragma solidity ^0.8.0;
/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts that want to implement ERC165 should inherit from this contract and
override {supportsInterface} to check
 * for the additional interface id that will be supported. For example:
 *
 * ```solidity
 * function supportsInterface(bytes4 interfaceId) public view virtual override
returns (bool) {
 *     return interfaceId == type(MyInterface).interfaceId ||
super.supportsInterface(interfaceId);
 * }
 * ```
 *
 * Alternatively, {ERC165Storage} provides an easier to use but more expensive
implementation.
 */
abstract contract ERC165 is IERC165 {
    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override
returns (bool) {
        return interfaceId == type(IERC165).interfaceId;
    }
}

// File contracts/utils/Strings.sol
// OpenZeppelin Contracts v4.4.1 (utils/Strings.sol)
pragma solidity ^0.8.0;
/**
 * @dev String operations.
 */
```

```
library Strings {
    bytes16 private constant _HEX_SYMBOLS = "0123456789abcdef";
    uint8 private constant _ADDRESS_LENGTH = 20;

    /**
     * @dev Converts a `uint256` to its ASCII `string` decimal representation.
     */
    function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        // https://github.com/oraclize/ethereum-
api/blob/b42146b063c7d6ee1358846c198246239e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
        }
        bytes memory buffer = new bytes(digits);
        while (value != 0) {
            digits -= 1;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
            value /= 10;
        }
        return string(buffer);
    }

    /**
     * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation.
     */
    function toHexString(uint256 value) internal pure returns (string memory) {
        if (value == 0) {
            return "0x00";
        }
        uint256 temp = value;
        uint256 length = 0;
        while (temp != 0) {
            length++;
            temp >>= 8;
        }
        return toHexString(value, length);
    }

    /**
     * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation
with fixed length.
```

```
     */
    function toHexString(uint256 value, uint256 length) internal pure returns (string
memory) {
        bytes memory buffer = new bytes(2 * length + 2);
        buffer[0] = "0";
        buffer[1] = "x";
        for (uint256 i = 2 * length + 1; i > 1; --i) {
            buffer[i] = _HEX_SYMBOLS[value & 0xf];
            value >>= 4;
        }
        require(value == 0, "Strings: hex length insufficient");
        return string(buffer);
    }

    /**
     * @dev Converts an `address` with fixed length of 20 bytes to its not
checksummed ASCII `string` hexadecimal representation.
     */
    function toHexString(address addr) internal pure returns (string memory) {
        return toHexString(uint256(uint160(addr)), _ADDRESS_LENGTH);
    }
}

// File contracts/access/AccessControl.sol
// OpenZeppelin Contracts (last updated v4.6.0) (access/AccessControl.sol)
pragma solidity ^0.8.0;
/**
 * @dev Contract module that allows children to implement role-based access
 * control mechanisms. This is a lightweight version that doesn't allow enumerating
role
 * members except through off-chain means by accessing the contract event logs. Some
 * applications may benefit from on-chain enumerability, for those cases see
 * {AccessControlEnumerable}.
 *
 * Roles are referred to by their `bytes32` identifier. These should be exposed
 * in the external API and be unique. The best way to achieve this is by
 * using `public constant` hash digests:
 *
 * ```
 * bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
 * ```
 *
 * Roles can be used to represent a set of permissions. To restrict access to a
 * function call, use {hasRole}:
 *
 * ```
 * function foo() public {
 *     require(hasRole(MY_ROLE, msg.sender));
 *     ...
```

```
 * }
 * ```
 *
 * Roles can be granted and revoked dynamically via the {grantRole} and
 * {revokeRole} functions. Each role has an associated admin role, and only
 * accounts that have a role's admin role can call {grantRole} and {revokeRole}.
 *
 * By default, the admin role for all roles is `DEFAULT_ADMIN_ROLE`, which means
 * that only accounts with this role will be able to grant or revoke other
 * roles. More complex role relationships can be created by using
 * {_setRoleAdmin}.
 *
 * WARNING: The `DEFAULT_ADMIN_ROLE` is also its own admin: it has permission to
 * grant and revoke this role. Extra precautions should be taken to secure
 * accounts that have been granted it.
 */
abstract contract AccessControl is Context, IAccessControl, ERC165 {
    struct RoleData {
        mapping(address => bool) members;
        bytes32 adminRole;
    }

    mapping(bytes32 => RoleData) private _roles;

    bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

    /**
     * @dev Modifier that checks that an account has a specific role. Reverts
     * with a standardized message including the required role.
     *
     * The format of the revert reason is given by the following regular expression:
     *
     *  /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
     *
     * _Available since v4.1._
     */
    modifier onlyRole(bytes32 role) {
        _checkRole(role);
        _;
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override
returns (bool) {
        return interfaceId == type(IAccessControl).interfaceId ||
super.supportsInterface(interfaceId);
    }
```

```
    /**
     * @dev Returns `true` if `account` has been granted `role`.
     */
    function hasRole(bytes32 role, address account) public view virtual override
returns (bool) {
        return _roles[role].members[account];
    }

    /**
     * @dev Revert with a standard message if `_msgSender()` is missing `role`.
     * Overriding this function changes the behavior of the {onlyRole} modifier.
     *
     * Format of the revert message is described in {_checkRole}.
     *
     * _Available since v4.6._
     */
    function _checkRole(bytes32 role) internal view virtual {
        _checkRole(role, _msgSender());
    }

    /**
     * @dev Revert with a standard message if `account` is missing `role`.
     *
     * The format of the revert reason is given by the following regular expression:
     *
     *  /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
     */
    function _checkRole(bytes32 role, address account) internal view virtual {
        if (!hasRole(role, account)) {
            revert(
                string(
                    abi.encodePacked(
                        "AccessControl: account ",
                        Strings.toHexString(uint160(account), 20),
                        " is missing role ",
                        Strings.toHexString(uint256(role), 32)
                    )
                )
            );
        }
    }

    /**
     * @dev Returns the admin role that controls `role`. See {grantRole} and
     * {revokeRole}.
     *
     * To change a role's admin, use {_setRoleAdmin}.
     */
```

```
    function getRoleAdmin(bytes32 role) public view virtual override returns
(bytes32) {
        return _roles[role].adminRole;
    }

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     *
     * May emit a {RoleGranted} event.
     */
    function grantRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
        _grantRole(role, account);
    }

    /**
     * @dev Revokes `role` from `account`.
     *
     * If `account` had been granted `role`, emits a {RoleRevoked} event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     *
     * May emit a {RoleRevoked} event.
     */
    function revokeRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
        _revokeRole(role, account);
    }

    /**
     * @dev Revokes `role` from the calling account.
     *
     * Roles are often managed via {grantRole} and {revokeRole}: this function's
     * purpose is to provide a mechanism for accounts to lose their privileges
     * if they are compromised (such as when a trusted device is misplaced).
     *
     * If the calling account had been revoked `role`, emits a {RoleRevoked}
     * event.
     *
     * Requirements:
```

```
     *
     * - the caller must be `account`.
     *
     * May emit a {RoleRevoked} event.
     */
    function renounceRole(bytes32 role, address account) public virtual override {
        require(account == _msgSender(), "AccessControl: can only renounce roles for
self");

        _revokeRole(role, account);
    }

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event. Note that unlike {grantRole}, this function doesn't perform any
     * checks on the calling account.
     *
     * May emit a {RoleGranted} event.
     *
     * [WARNING]
     * ====
     * This function should only be called from the constructor when setting
     * up the initial roles for the system.
     *
     * Using this function in any other way is effectively circumventing the admin
     * system imposed by {AccessControl}.
     * ====
     *
     * NOTE: This function is deprecated in favor of {_grantRole}.
     */
    function _setupRole(bytes32 role, address account) internal virtual {
        _grantRole(role, account);
    }

    /**
     * @dev Sets `adminRole` as ``role``'s admin role.
     *
     * Emits a {RoleAdminChanged} event.
     */
    function _setRoleAdmin(bytes32 role, bytes32 adminRole) internal virtual {
        bytes32 previousAdminRole = getRoleAdmin(role);
        _roles[role].adminRole = adminRole;
        emit RoleAdminChanged(role, previousAdminRole, adminRole);
    }

    /**
     * @dev Grants `role` to `account`.
```

```
     *
     * Internal function without access restriction.
     *
     * May emit a {RoleGranted} event.
     */
    function _grantRole(bytes32 role, address account) internal virtual {
        if (!hasRole(role, account)) {
            _roles[role].members[account] = true;
            emit RoleGranted(role, account, _msgSender());
        }
    }

    /**
     * @dev Revokes `role` from `account`.
     *
     * Internal function without access restriction.
     *
     * May emit a {RoleRevoked} event.
     */
    function _revokeRole(bytes32 role, address account) internal virtual {
        if (hasRole(role, account)) {
            _roles[role].members[account] = false;
            emit RoleRevoked(role, account, _msgSender());
        }
    }
}

// File contracts/access/IAccessControlEnumerable.sol
// OpenZeppelin Contracts v4.4.1 (access/IAccessControlEnumerable.sol)
pragma solidity ^0.8.0;
/**
 * @dev External interface of AccessControlEnumerable declared to support ERC165
detection.
 */
interface IAccessControlEnumerable is IAccessControl {
    /**
     * @dev Returns one of the accounts that have `role`. `index` must be a
     * value between 0 and {getRoleMemberCount}, non-inclusive.
     *
     * Role bearers are not sorted in any particular way, and their ordering may
     * change at any point.
     *
     * WARNING: When using {getRoleMember} and {getRoleMemberCount}, make sure
     * you perform all queries on the same block. See the following
     * https://forum.openzeppelin.com/t/iterating-over-elements-on-enumerableset-in-
openzeppelin-contracts/2296[forum post]
     * for more information.
     */
    function getRoleMember(bytes32 role, uint256 index) external view returns
```

```
(address);

    /**
     * @dev Returns the number of accounts that have `role`. Can be used
     * together with {getRoleMember} to enumerate all bearers of a role.
     */
    function getRoleMemberCount(bytes32 role) external view returns (uint256);
}

// File contracts/utils/structs/EnumerableSet.sol
// OpenZeppelin Contracts (last updated v4.6.0) (utils/structs/EnumerableSet.sol)
pragma solidity ^0.8.0;

/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 * ```
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 * ```
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 *
 * [WARNING]
 * ====
 *  Trying to delete such a structure from storage will likely result in data
corruption, rendering the structure unusable.
 *  See https://github.com/ethereum/solidity/pull/11843[ethereum/solidity#11843] for
more info.
 *
 *  In order to clean an EnumerableSet, you can either remove all elements one by one
or create a fresh instance using an array of EnumerableSet.
 * ====
 */
library EnumerableSet {
```

```
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;
        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping(bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the
same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) {
            // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the
```

```
element to delete with the last one in
            // the array, and then remove the last element (sometimes called as 'swap
and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            if (lastIndex != toDeleteIndex) {
                bytes32 lastValue = set._values[lastIndex];

                // Move the last value to the index where the value to delete is
                set._values[toDeleteIndex] = lastValue;
                // Update the index for the moved value
                set._indexes[lastValue] = valueIndex; // Replace lastValue's index to
valueIndex
            }

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
```

```
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        return set._values[index];
    }

    /**
     * @dev Return the entire set in an array
     *
     * WARNING: This operation will copy the entire storage to memory, which can be
quite expensive. This is designed
     * to mostly be used by view accessors that are queried without any gas fees.
Developers should keep in mind that
     * this function has an unbounded cost, and using it as part of a state-changing
function may render the function
     * uncallable if the set grows to a point where copying to memory consumes too
much gas to fit in a block.
     */
    function _values(Set storage set) private view returns (bytes32[] memory) {
        return set._values;
    }

    // Bytes32Set

    struct Bytes32Set {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _add(set._inner, value);
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _remove(set._inner, value);
```

```
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(Bytes32Set storage set, bytes32 value) internal view returns
(bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(Bytes32Set storage set, uint256 index) internal view returns
(bytes32) {
        return _at(set._inner, index);
    }

    /**
     * @dev Return the entire set in an array
     *
     * WARNING: This operation will copy the entire storage to memory, which can be
quite expensive. This is designed
     * to mostly be used by view accessors that are queried without any gas fees.
Developers should keep in mind that
     * this function has an unbounded cost, and using it as part of a state-changing
function may render the function
     * uncallable if the set grows to a point where copying to memory consumes too
much gas to fit in a block.
     */
    function values(Bytes32Set storage set) internal view returns (bytes32[] memory)
{
        return _values(set._inner);
    }
```

```
    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns
(bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
```

```
     * - `index` must be strictly less than {length}.
     */
    function at(AddressSet storage set, uint256 index) internal view returns
(address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }

    /**
     * @dev Return the entire set in an array
     *
     * WARNING: This operation will copy the entire storage to memory, which can be
quite expensive. This is designed
     * to mostly be used by view accessors that are queried without any gas fees.
Developers should keep in mind that
     * this function has an unbounded cost, and using it as part of a state-changing
function may render the function
     * uncallable if the set grows to a point where copying to memory consumes too
much gas to fit in a block.
     */
    function values(AddressSet storage set) internal view returns (address[] memory)
{
        bytes32[] memory store = _values(set._inner);
        address[] memory result;

        assembly {
            result := store
        }

        return result;
    }

    // UintSet

    struct UintSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
```

```
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns
(bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }

    /**
     * @dev Return the entire set in an array
     *
     * WARNING: This operation will copy the entire storage to memory, which can be
quite expensive. This is designed
     * to mostly be used by view accessors that are queried without any gas fees.
Developers should keep in mind that
     * this function has an unbounded cost, and using it as part of a state-changing
function may render the function
     * uncallable if the set grows to a point where copying to memory consumes too
much gas to fit in a block.
     */
```

```
    function values(UintSet storage set) internal view returns (uint256[] memory) {
        bytes32[] memory store = _values(set._inner);
        uint256[] memory result;

        assembly {
            result := store
        }

        return result;
    }
}

// File contracts/access/AccessControlEnumerable.sol
// OpenZeppelin Contracts (last updated v4.5.0) (access/AccessControlEnumerable.sol)
pragma solidity ^0.8.0;
/**
 * @dev Extension of {AccessControl} that allows enumerating the members of each
role.
 */
abstract contract AccessControlEnumerable is IAccessControlEnumerable, AccessControl
{
    using EnumerableSet for EnumerableSet.AddressSet;

    mapping(bytes32 => EnumerableSet.AddressSet) private _roleMembers;

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override
returns (bool) {
        return interfaceId == type(IAccessControlEnumerable).interfaceId ||
super.supportsInterface(interfaceId);
    }

    /**
     * @dev Returns one of the accounts that have `role`. `index` must be a
     * value between 0 and {getRoleMemberCount}, non-inclusive.
     *
     * Role bearers are not sorted in any particular way, and their ordering may
     * change at any point.
     *
     * WARNING: When using {getRoleMember} and {getRoleMemberCount}, make sure
     * you perform all queries on the same block. See the following
     * https://forum.openzeppelin.com/t/iterating-over-elements-on-enumerableset-in-
openzeppelin-contracts/2296[forum post]
     * for more information.
     */
    function getRoleMember(bytes32 role, uint256 index) public view virtual override
returns (address) {
```

```
        return _roleMembers[role].at(index);
    }

    /**
     * @dev Returns the number of accounts that have `role`. Can be used
     * together with {getRoleMember} to enumerate all bearers of a role.
     */
    function getRoleMemberCount(bytes32 role) public view virtual override returns
(uint256) {
        return _roleMembers[role].length();
    }

    /**
     * @dev Overload {_grantRole} to track enumerable memberships
     */
    function _grantRole(bytes32 role, address account) internal virtual override {
        super._grantRole(role, account);
        _roleMembers[role].add(account);
    }

    /**
     * @dev Overload {_revokeRole} to track enumerable memberships
     */
    function _revokeRole(bytes32 role, address account) internal virtual override {
        super._revokeRole(role, account);
        _roleMembers[role].remove(account);
    }
}

// File contracts/access/Ownable.sol
// OpenZeppelin Contracts v4.4.1 (access/Ownable.sol)
pragma solidity ^0.8.0;
/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

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

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

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

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

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

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
```

```
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}

// File contracts/KIP/token/KIP7/extensions/IKIP7Pausable.sol
// Klaytn Contract Library v1.0.0 (KIP/token/KIP7/extensions/IKIP7Pausable.sol)
pragma solidity ^0.8.0;
/**
 * @dev Pausing extension of the KIP7 standard as defined in the KIP.
 * See https://kips.klaytn.com/KIPs/kip-7#pausing-extension
 */
interface IKIP7Pausable {
    /**
     * @dev Returns true if the contract is paused, false otherwise
     */
    function paused() external view returns (bool);
    /**
     * @dev Pause any function which triggers {KIP7-_beforeTokenTransfer}
     *
     * Emits a {Paused} event.
     *
     * Requirements:
     *
     * - caller must have the {KIP7Pausable-PAUSER_ROLE}
     */
    function pause() external;
    /**
     * @dev Resume normal function from the paused state
     *
     * Emits a {Unpaused} event.
     *
     * Requirements:
     *
     * - caller must have the {KIP7Pausable-PAUSER_ROLE}
     */
    function unpause() external;
    /**
     * @dev Check if `account` has the assigned Pauser role via {AccessControl-
hasRole}
     */
    function isPauser(address account) external view returns (bool);
    /**
     * @dev Assign the Pauser role to `account` via {AccessControl-grantRole}
     *
     * Emits a {RoleGranted} event
     *
```

```
     * Requirements:
     *
     * - caller must have the {AccessControl-DEFAULT_ADMIN_ROLE}
     */
    function addPauser(address _account) external;
    /**
     * @dev Renounce the Pauser role of the caller via {AccessControl-renounceRole}
     *
     * Emits a {RoleRevoked} event
     */
    function renouncePauser() external;
}

// File contracts/KIP/interfaces/IKIP7Pausable.sol
// Klaytn Contract Library v1.0.0 (KIP/interfaces/IKIP7Pausable.sol)
pragma solidity ^0.8.0;
// File contracts/KIP/token/KIP7/IKIP7.sol
// Klaytn Contract Library v1.0.0 (KIP/token/KIP7/IKIP7.sol)
// Based on OpenZeppelin Contracts v4.5.0 (token/ERC20/IERC20.sol)
// https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.5.0
pragma solidity ^0.8.0;
/**
 * @dev Interface of the KIP7 standard as defined in the KIP.
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
interface IKIP7 {
    /**
     * @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 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 `to`.
     *
```

```
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address to, 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}, {transferFrom}, or {safeTransferFrom} 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 `from` to `to` 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 from,
        address to,
        uint256 amount
    ) external returns (bool);
    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`
     * and passes `data` for {IKIP7Receiver-onKIP7Received} handler logic.
     *
     * Emits a {Transfer} event.
```

```
     */
    function safeTransfer(
        address recipient,
        uint256 amount,
        bytes memory data
    ) external;
    /**
     * @dev  Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Emits a {Transfer} event.
     */
    function safeTransfer(address recipient, uint256 amount) external;
    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the {allowance}
mechanism
     * and passes `data` for {IKIP7Receiver-onKIP7Received} handler logic.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
        address sender,
        address recipient,
        uint256 amount,
        bytes memory data
    ) external;
    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the {allowance}
mechanism.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external;
}
// File contracts/KIP/token/KIP7/extensions/IKIP7Metadata.sol
// Klaytn Contract Library v1.0.0 (KIP/token/KIP7/extensions/IKIP7Metadata.sol)
// Based on OpenZeppelin Contracts v4.5.0 (token/ERC20/extensions/IERC20Metadata.sol)
// https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.5.0
pragma solidity ^0.8.0;
/**
 * @dev Extension of {KIP7} which exposes metadata functions.
 * See https://kips.klaytn.com/KIPs/kip-7#metadata-extension
 */
interface IKIP7Metadata is IKIP7 {
    /**
     * @dev Returns the name of the token.
```

```
     */
    function name() external view returns (string memory);

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

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

// File contracts/KIP/token/KIP7/IKIP7Receiver.sol
// Klaytn Contract Library v1.0.0 (KIP/token/KIP7/IKIP7Receiver.sol)
// Based on OpenZeppelin Contracts v4.5.0 (token/ERC20/IERC20Receiver.sol)
// https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.5.0
pragma solidity ^0.8.0;

interface IKIP7Receiver {
    /**
     * @dev Whenever an {IKIP7} `amount` is transferred to this contract via {IKIP7-
safeTransfer}
     * or {IKIP7-safeTransferFrom} by `operator` from `from`, this function is
called.
     *
     * {onKIP7Received} must return its Solidity selector to confirm the token
transfer.
     * If any other value is returned or the interface is not implemented by the
recipient, the transfer will be reverted.
     *
     * The selector can be obtained in Solidity with
`IKIP7Receiver.onKIP7Received.selector`.
     */
    function onKIP7Received(
        address operator,
        address from,
        uint256 amount,
        bytes calldata _data
    ) external returns (bytes4);
}
// File contracts/KIP/utils/introspection/IKIP13.sol
// Klaytn Contract Library v1.0.0 (KIP/utils/introspection/IKIP13.sol)
// Based on OpenZeppelin Contracts v4.5.0 (utils/introspection/IERC165.sol)
// https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.5.0
pragma solidity ^0.8.0;
/**
 * @dev Interface of the KIP13 standard as defined in the KIP.
```

```
 *
 * See - http://kips.klaytn.com/KIPs/kip-13-interface_query_standard
 */
interface IKIP13 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`.
     * See - http://kips.klaytn.com/KIPs/kip-13-interface_query_standard#how-
interface-identifiers-are-defined
     * to learn more about how these ids are created.
     *
     * Requirements:
     *
     * - implementation of this function call must use less than 30 000 gas
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

// File contracts/KIP/interfaces/IKIP13.sol
// Klaytn Contract Library v1.0.0 (KIP/interfaces/IKIP13.sol)
pragma solidity ^0.8.0;
// File contracts/KIP/utils/introspection/KIP13.sol
// Klaytn Contract Library v1.0.0 (KIP/utils/introspection/KIP13.sol)
// Based on OpenZeppelin Contracts v4.5.0 (utils/introspection/ERC165.sol)
// https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.5.0
pragma solidity ^0.8.0;
/**
 * @dev Implementation of the {IKIP13} interface.
 *
 * Contracts that want to implement KIP13 should inherit from this contract and
override {supportsInterface} to check
 * for the additional interface id that will be supported. For example:
 *
 * ```solidity
 * function supportsInterface(bytes4 interfaceId) public view virtual override
returns (bool) {
 *     return interfaceId == type(MyInterface).interfaceId ||
super.supportsInterface(interfaceId);
 * }
 * ```
 *
 * Alternatively, {KIP13Storage} provides an easier to use but more expensive
implementation.
 */
abstract contract KIP13 is IKIP13 {
    /**
     * @dev See {IKIP13-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override
```

```
returns (bool) {
        return interfaceId == type(IKIP13).interfaceId;
    }
}
// File contracts/utils/Address.sol
// OpenZeppelin Contracts (last updated v4.5.0) (utils/Address.sol)
pragma solidity ^0.8.1;
/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     *
     * [IMPORTANT]
     * ====
     * You shouldn't rely on `isContract` to protect against flash loan attacks!
     *
     * Preventing calls from contracts is highly discouraged. It breaks
composability, breaks support for smart wallets
     * like Gnosis Safe, and does not provide security since it can be circumvented
by calling from a contract
     * constructor.
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize/address.code.length, which returns 0
        // for contracts in construction, since the code is only stored at the end
        // of the constructor execution.

        return account.code.length > 0;
    }
    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
```

```
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-
now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-
the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may have
reverted");
    }
    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain `call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?
highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes
memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but
with
     * `errorMessage` as a fallback revert reason when `target` reverts.
```

```
     *
     * _Available since v3.1._
     */
    function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, errorMessage);
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call
with value failed");
    }
    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}
[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for
call");
        require(isContract(target), "Address: call to non-contract");

        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
```

```
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view
returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call
failed");
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}
[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns
(bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call
failed");
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}
[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
```

```
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    /**
     * @dev Tool to verifies that a low level call was successful, and revert if it
wasn't, either by bubbling the
     * revert reason using the provided one.
     *
     * _Available since v4.3._
     */
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via
assembly

                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

// File contracts/KIP/token/KIP7/KIP7.sol
// Klaytn Contract Library v1.0.0 (KIP/token/KIP7/KIP7.sol)
// Based on OpenZeppelin Contracts v4.5.0 (token/ERC20/ERC20.sol)
// https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.5.0
pragma solidity ^0.8.0;
/**
 * @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 {KIP7PresetMinterPauser}.
```

```
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of 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 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 {IKIP7-approve}.
 *
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract KIP7 is Context, KIP13, IKIP7, IKIP7Metadata {
    using Address for address;

    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string internal _name;
    string internal _symbol;
    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }
    /**
     * @dev See {IKIP13-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual
override(KIP13) returns (bool) {
        return
            interfaceId == type(IKIP7).interfaceId ||
            interfaceId == type(IKIP7Metadata).interfaceId ||
```

```
            KIP13.supportsInterface(interfaceId);
    }
    /**
     * @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 {KIP7} uses, unless this function is
     * overridden;
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IKIP7-balanceOf} and {IKIP7-transfer}.
     */
    function decimals() public view virtual override returns (uint8) {
        return 18;
    }
    /**
     * @dev See {IKIP7-totalSupply}.
     */
    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }
    /**
     * @dev See {IKIP7-balanceOf}.
     */
    function balanceOf(address account) public view virtual override returns
(uint256) {
        return _balances[account];
    }
    /**
     * @dev See {IKIP7-transfer}.
     *
     * Requirements:
     *
```

```
     * - `to` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address to, uint256 amount) public virtual override returns
(bool) {
        address owner = _msgSender();
        _transfer(owner, to, amount);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }
    /**
     * @dev See {IKIP7-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override
returns (uint256) {
        return _allowances[owner][spender];
    }
    /**
     * @dev See {IKIP7-approve}.
     *
     * NOTE: If `amount` is the maximum `uint256`, the allowance is not updated on
     * `transferFrom`. This is semantically equivalent to an infinite approval.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns
(bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        //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 EIP. See the note at the beginning of {KIP7}.
     *
     * NOTE: Does not update the allowance if the current allowance
     * is the maximum `uint256`.
     *
     * Requirements:
     *
     * - `from` and `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     * - the caller must have allowance for ``from``'s tokens of at least
     * `amount`.
```

```
     */
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }
    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IKIP7-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) {
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, 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 {IKIP7-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) {
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spender);
        require(currentAllowance >= subtractedValue, "KIP7: decreased allowance below
```

```
zero");
        unchecked {
            _approve(owner, spender, currentAllowance - subtractedValue);
        }

        return true;
    }
    /**
     * @dev See {IKIP7-safeTransfer}.
     *
     * Emits a {Transfer} event
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     * - if `recipient` is a smart contract, it must implement {IKIP7Receiver}
     */
    function safeTransfer(address recipient, uint256 amount) public virtual override
{
        address owner = _msgSender();
        _safeTransfer(owner, recipient, amount, "");
    }
    /**
     * @dev Same as {xref-KIP7-safeTransfer-address-uint256-}[`safeTransfer`], with
an additional `_data` parameter which is
     * forwarded in {IKIP7Receiver-onKIP7Received} to contract recipients.
     *
     * Emits a {Transfer} event
     */
    function safeTransfer(
        address recipient,
        uint256 amount,
        bytes memory _data
    ) public virtual override {
        address owner = _msgSender();
        _safeTransfer(owner, recipient, amount, _data);
    }
    /**
     * @dev See {IKIP7-safeTransferFrom}.
     *
     * Emits a {Transfer} event
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the KIP. See the note at the beginning of {KIP7}.
     *
     * NOTE: Does not update the allowance if the current allowance
     * is the maximum `uint256`.
     *
```

```
     * 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`.
     * - if `recipient` is a smart contract, it must implement {IKIP7Receiver}
     */
    function safeTransferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public virtual override {
        address spender = _msgSender();
        _spendAllowance(sender, spender, amount);
        _safeTransfer(sender, recipient, amount, "");
    }
    /**
     * @dev Same as {xref-KIP7-safeTransferFrom-address-uint256-}
[`safeTransferFrom`], with an additional `_data` parameter which is
     * forwarded in {IKIP7Receiver-onKIP7Received} to contract recipients.
     *
     * Emits a {Transfer} event
     *
     * Emits an {Approval} event indicating the updated allowance.
     */
    function safeTransferFrom(
        address sender,
        address recipient,
        uint256 amount,
        bytes memory _data
    ) public virtual override {
        address spender = _msgSender();
        _spendAllowance(sender, spender, amount);
        _safeTransfer(sender, recipient, amount, _data);
    }
    /**
     * @dev Safely transfers `amout` token from `from` to `to`, checking first that
contract recipients
     * are aware of the KIP7 protocol to prevent tokens from being forever locked.
     *
     * `_data` is additional data, it has no specified format and it is sent in call
to `to` to be used
     * for additional KIP7 receiver handler logic
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
```

```
     * - `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     * - if `to` is a contract it must implement {IKIP7Receiver} and
     * - If `to` refers to a smart contract, it must implement {IKIP7Receiver}, which
is called upon
     *   a safe transfer.
     */
    function _safeTransfer(
        address from,
        address to,
        uint256 amount,
        bytes memory _data
    ) internal virtual {
        _transfer(from, to, amount);
        require(_checkOnKIP7Received(from, to, amount, _data), "KIP7: transfer to non
IKIP7Receiver implementer");
    }
    /**
     * @dev Moves `amount` of tokens from `sender` to `recipient`.
     *
     * This internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     */
    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {
        require(from != 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(to != address(0), "KIP7: transfer to the zero address");

        _beforeTokenTransfer(from, to, amount);

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "KIP7: transfer amount exceeds balance");
        unchecked {
            _balances[from] = fromBalance - amount;
        }
        _balances[to] += amount;
```

```
        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }
    /**
     * @dev Safely mints `amount` and transfers it to `account`.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - If `account` refers to a smart contract, it must implement {IKIP7Receiver-
onKIP7Received},
     *   which is called upon a safe mint.
     *
     * Emits a {Transfer} event.
     */
    function _safeMint(address account, uint256 amount) internal virtual {
        _safeMint(account, amount, "");
    }
    /**
     * @dev Same as {xref-KIP7-_safeMint-address-uint256-}[`_safeMint`], with an
additional `_data` parameter which is
     * forwarded in {IKIP7Receiver-onKIP7Received} to contract recipients.
     */
    function _safeMint(
        address account,
        uint256 amount,
        bytes memory _data
    ) internal virtual {
        _mint(account, amount);
        require(
            _checkOnKIP7Received(address(0), account, amount, _data),
            "KIP7: transfer to non IKIP7Receiver implementer"
        );
    }
    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * WARNING: Usage of this method is discouraged, use {_safeMint} whenever
possible
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
```

```
        require(account != address(0), "KIP7: mint to the zero address");

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

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

        _afterTokenTransfer(address(0), account, amount);
    }
    /**
     * @dev Internal function to invoke {IKIP7Receiver-onKIP7Received} on a target
address.
     * The call is not executed if the target address is not a contract.
     *
     * @param from address representing the previous owner of transfer token amount
     * @param to target address that will receive the tokens
     * @param amount uint256 value to be transferred
     * @param _data bytes optional data to send along with the call
     * @return bool whether the call correctly returned the expected magic value
     */
    function _checkOnKIP7Received(
        address from,
        address to,
        uint256 amount,
        bytes memory _data
    ) private returns (bool) {
        if (to.isContract()) {
            try IKIP7Receiver(to).onKIP7Received(_msgSender(), from, amount, _data)
returns (bytes4 retval) {
                return retval == IKIP7Receiver.onKIP7Received.selector;
            } catch (bytes memory reason) {
                if (reason.length == 0) {
                    revert("KIP7: transfer to non KIP7Receiver implementer");
                } else {
                    assembly {
                        revert(add(32, reason), mload(reason))
                    }
                }
            }
        } else {
            return true;
        }
    }
    /**
     * @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), "KIP7: burn from the zero address");

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

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "KIP7: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
        }
        _totalSupply -= amount;

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

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

```
     * @dev Updates `owner` s allowance for `spender` based on spent `amount`.
     *
     * Does not update the allowance amount in case of infinite allowance.
     * Revert if not enough allowance is available.
     *
     * Might emit an {Approval} event.
     */
    function _spendAllowance(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "KIP7: insufficient allowance");
            unchecked {
                _approve(owner, spender, currentAllowance - amount);
            }
        }
    }
    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-
hooks[Using Hooks].
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
    /**
     * @dev Hook that is called after any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been minted for `to`.
```

```
     * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-
hooks[Using Hooks].
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}

// File contracts/security/Pausable.sol
// OpenZeppelin Contracts v4.4.1 (security/Pausable.sol)
pragma solidity ^0.8.0;
/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
abstract contract Pausable is Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account);

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

    bool private _paused;

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

```
    }
    /**
     * @dev Modifier to make a function callable only when the contract is not
paused.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
        require(!paused(), "Pausable: paused");
        _;
    }
    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    modifier whenPaused() {
        require(paused(), "Pausable: not paused");
        _;
    }
    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    function _pause() internal virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }
    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function _unpause() internal virtual whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}

// File contracts/KIP/token/KIP7/extensions/KIP7Pausable.sol
```

```
// Klaytn Contract Library v1.0.0 (KIP/token/KIP7/extensions/KIP7Pausable.sol)
// Based on OpenZeppelin Contracts v4.5.0 (token/ERC20/extensions/ERC20Pausable.sol)
// https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.5.0
pragma solidity ^0.8.0;
/**
 * @dev Extension of KIP7 that supports permissioned pausable token transfers,
minting and burning.
 *
 * Useful for scenarios such as preventing transfers until the end of an evaluation
 * period, or having an emergency switch for freezing all token transfers in the
 * event of a large bug.
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
abstract contract KIP7Pausable is KIP7, AccessControlEnumerable, Pausable,
IKIP7Pausable {
    bytes32 public constant PAUSER_ROLE = keccak256("KIP7_PAUSER_ROLE");

    /**
     * @dev Returns true if `interfaceId` is implemented and false otherwise
     *
     * See {IKIP13} and {IERC165}.
     */
    function supportsInterface(bytes4 interfaceId)
        public
        view
        virtual
        override(KIP7, AccessControlEnumerable)
        returns (bool)
    {
        return
            interfaceId == type(IKIP7Pausable).interfaceId ||
            KIP7.supportsInterface(interfaceId) ||
            AccessControlEnumerable.supportsInterface(interfaceId);
    }
    /**
     * @dev See {IKIP7Pausable-pause}
     *
     * Emits a {Paused} event.
     *
     * Requirements:
     *
     * - caller must have the {PAUSER_ROLE}
     */
    //SlowMist// Suspending all transactions upon major abnormalities is a
recommended approach
    function pause() public virtual override {
        require(hasRole(PAUSER_ROLE, _msgSender()), "KIP7PresetMinterPauser: must
have pauser role to pause");
        _pause();
```

```
    }
    /**
     * @dev See {IKIP7Pausable-unpause}
     *
     * Emits a {Unpaused} event.
     *
     * Requirements:
     *
     * - caller must have the {PAUSER_ROLE}
     */
    function unpause() public virtual override {
        require(hasRole(PAUSER_ROLE, _msgSender()), "KIP7PresetMinterPauser: must
have pauser role to unpause");
        _unpause();
    }

    /**
     * @dev Returns true if the contract is paused, false otherwise
     */
    function paused() public view override(IKIP7Pausable, Pausable) returns (bool) {
        return super.paused();
    }

    /**
     * @dev Check if `account` has the assigned Pauser role via {AccessControl-
hasRole}
     */
    function isPauser(address account) public view override returns (bool) {
        return hasRole(PAUSER_ROLE, account);
    }

    /**
     * @dev See {IKIP7Pausable-addPauser}
     *
     * Emits a {RoleGranted} event
     *
     * Requirements:
     *
     * - caller must have the {AccessControl-DEFAULT_ADMIN_ROLE}
     */
    function addPauser(address account) public override onlyRole(DEFAULT_ADMIN_ROLE)
{
        grantRole(PAUSER_ROLE, account);
    }

    /**
     * @dev Renounce the Pauser role of the caller via {AccessControl-renounceRole}
     *
     * Emits a {RoleRevoked} event
```

```
     */
    function renouncePauser() public override {
        renounceRole(PAUSER_ROLE, _msgSender());
    }

    /**
     * @dev See {KIP7-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - the contract must not be paused.
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        require(!paused(), "KIP7Pausable: token transfer while paused");
    }
}
// File contracts/Favor.sol
pragma solidity ^0.8.0;

contract Favor is KIP7Pausable, Ownable {
  uint256 public constant FAVOR_SUPPLY_LIMIT = 3e8 ether; // 300,000,000

  constructor(
    string memory name,
    string memory symbol
  ) KIP7(name, symbol) Ownable() {
    _setupRole(PAUSER_ROLE, _msgSender());
    _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
    _mint(_msgSender(), FAVOR_SUPPLY_LIMIT);
  }
  //SlowMist// Missing the event log
  function setName(string memory name) public onlyOwner {
    _name = name;
  }
  //SlowMist// Missing the event log
  function setSymbol(string memory symbol) public onlyOwner {
    _symbol = symbol;
  }
}
```

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