



Smart Contract Security Audit Report



The SlowMist Security Team received the team's application for smart contract security audit of the MEVerse on 2022.01.19. The following are the details and results of this smart contract security audit:

Token Name :

MEVerse

The contract address :

<https://etherscan.io/address/0xb3Cb8d5AEFF0F4d1f432f353309f47b885E404E3>

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	Some Risks
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed

Audit Result : Low Risk

Audit Number : 0X002201210001

Audit Date : 2022.01.19 - 2022.01.21

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their own tokens through the burn function. SafeMath security module is used, which is a recommended approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following issue:

1. The owner role can add a minter role, and the minter role can mint tokens arbitrarily through the mint function and there is no upper limit of the minted amount of tokens which can increase token issuance and cause depreciation.

After communicating with the project team, the project team expresses that they left the mint function in purpose because of their cross-chain bridge tokenomics among ETH-BSC-Polygon-Klaytn. And they plan to decentralize the bridge system this year.

The source code:

ERC20PresetMinterPauser.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/presets/ERC20PresetMinterPauser.sol)

/*
forked from https://github.com/OpenZeppelin/openzeppelin-
contracts/tree/783ac759a902a7b4a218c2d026a77e6a26b6c42d
```

```

replace path
add init supply on constructor
*/
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;

import "./util/ERC20.sol";
import "./util/ERC20Burnable.sol";
import "./util/ERC20Pausable.sol";
import "./util/AccessControlEnumerable.sol";
import "./util/Context.sol";

/**
 * @dev {ERC20} token, including:
 *
 * - ability for holders to burn (destroy) their tokens
 * - a minter role that allows for token minting (creation)
 * - a pauser role that allows to stop all token transfers
 *
 * This contract uses {AccessControl} to lock permissioned functions using the
 * different roles - head to its documentation for details.
 *
 * The account that deploys the contract will be granted the minter and pauser
 * roles, as well as the default admin role, which will let it grant both minter
 * and pauser roles to other accounts.
 *
 * _Deprecated in favor of https://wizard.openzeppelin.com/\[Contracts Wizard\].\_
 */
contract ERC20PresetMinterPauser is Context, AccessControlEnumerable, ERC20Burnable,
ERC20Pausable {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");

    /**
     * @dev Grants `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE` and `PAUSER_ROLE` to the
     * account that deploys the contract.
     *
     * See {ERC20-constructor}.
     */
    constructor(string memory name, string memory symbol) ERC20(name, symbol) {
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());

        _setupRole(MINTER_ROLE, _msgSender());
        _setupRole(PAUSER_ROLE, _msgSender());
    }
}

```

```

        // init supply 2 billion
        mint(_msgSender(), 2000000000000000000000000);
    }

    /**
     * @dev Creates `amount` new tokens for `to`.
     *
     * See {ERC20-_mint}.
     *
     * Requirements:
     *
     * - the caller must have the `MINTER_ROLE`.
     */
    //SlowMist// The admin role can add a new minter role and the minter role can
mint tokens arbitrarily through the mint function and there is no upper limit of the
minted amount of tokens
    function mint(address to, uint256 amount) public virtual {
        require(hasRole(MINTER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must
have minter role to mint");
        _mint(to, amount);
    }

    /**
     * @dev Pauses all token transfers.
     *
     * See {ERC20Pausable} and {Pausable-_pause}.
     *
     * Requirements:
     *
     * - the caller must have the `PAUSER_ROLE`.
     */
    //SlowMist// Suspending all transactions upon major abnormalities is a
recommended approach
    function pause() public virtual {
        require(hasRole(PAUSER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must
have pauser role to pause");
        _pause();
    }

    /**
     * @dev Unpauses all token transfers.
     *
     * See {ERC20Pausable} and {Pausable-_unpause}.

```

```

*
* Requirements:
*
* - the caller must have the `PAUSER_ROLE`.
*/
function unpause() public virtual {
    require(hasRole(PAUSER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must
have pauser role to unpause");
    _unpause();
}

function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override(ERC20, ERC20Pausable) {
    super._beforeTokenTransfer(from, to, amount);
}
}
    
```

ERC20.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/ERC20.sol)
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;

import "./IERC20.sol";
import "./IERC20Metadata.sol";
import "./Context.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
    
```

```

*
* We have followed general OpenZeppelin Contracts guidelines: functions revert
* instead returning `false` on failure. This behavior is nonetheless
* conventional and does not conflict with the expectations of ERC20
* applications.
*
* Additionally, an {Approval} event is emitted on calls to {transferFrom}.
* This allows applications to reconstruct the allowance for all accounts just
* by listening to said events. Other implementations of the EIP may not emit
* these events, as it isn't required by the specification.
*
* Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
* functions have been added to mitigate the well-known issues around setting
* allowances. See {IERC20-approve}.
*/
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The 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 Returns the name of the token.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

```

```

}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view virtual override returns (string memory) {
    return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
 * be displayed to a user as `5.05` ( $505 / 10 ** 2$ ).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei. This is the value {ERC20} uses, unless this function is
 * overridden;
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
 * {IERC20-balanceOf} and {IERC20-transfer}.
 */
function decimals() public view virtual override returns (uint8) {
    return 18;
}

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

/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(address account) public view virtual override returns
(uint256) {
    return _balances[account];
}

/**
 * @dev See {IERC20-transfer}.

```

```

*
* Requirements:
*
* - `recipient` cannot be the zero address.
* - the caller must have a balance of at least `amount`.
*/
function transfer(address recipient, uint256 amount) public virtual override
returns (bool) {
    _transfer(_msgSender(), recipient, amount);
    //SlowMist// The return value conforms to the EIP20 specification
    return true;
}

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

/**
 * @dev See {IERC20-approve}.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function approve(address spender, uint256 amount) public virtual override returns
(bool) {
    _approve(_msgSender(), spender, amount);
    //SlowMist// The return value conforms to the EIP20 specification
    return true;
}

/**
 * @dev See {IERC20-transferFrom}.
 *
 * Emits an {Approval} event indicating the updated allowance. This is not
 * required by the EIP. See the note at the beginning of {ERC20}.
 *
 * Requirements:
 *
 * - `sender` and `recipient` cannot be the zero address.

```

```

* - `sender` must have a balance of at least `amount`.
* - the caller must have allowance for ``sender``'s tokens of at least
* `amount`.
*/
function transferFrom(
    address sender,
    address recipient,
    uint256 amount
) public virtual override returns (bool) {
    _transfer(sender, recipient, amount);

    uint256 currentAllowance = _allowances[sender][_msgSender()];
    require(currentAllowance >= amount, "ERC20: transfer amount exceeds
allowance");
    unchecked {
        _approve(sender, _msgSender(), currentAllowance - amount);
    }
    //SlowMist// The return value conforms to the EIP20 specification
    return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {IERC20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(address spender, uint256 addedValue) public virtual
returns (bool) {
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender] +
addedValue);
    return true;
}

/**
 * @dev Atomically decreases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for

```

```

* problems described in {IERC20-approve}.
*
* Emits an {Approval} event indicating the updated allowance.
*
* Requirements:
*
* - `spender` cannot be the zero address.
* - `spender` must have allowance for the caller of at least
* `subtractedValue`.
*/
function decreaseAllowance(address spender, uint256 subtractedValue) public
virtual returns (bool) {
    uint256 currentAllowance = _allowances[_msgSender()][spender];
    require(currentAllowance >= subtractedValue, "ERC20: decreased allowance
below zero");
    unchecked {
        _approve(_msgSender(), spender, currentAllowance - subtractedValue);
    }

    return true;
}

/**
* @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:
*
* - `sender` cannot be the zero address.
* - `recipient` cannot be the zero address.
* - `sender` must have a balance of at least `amount`.
*/
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual {
    //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
    require(sender != address(0), "ERC20: transfer from the zero address");

```

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

_beforeTokenTransfer(sender, recipient, amount);

uint256 senderBalance = _balances[sender];
require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
unchecked {
    _balances[sender] = senderBalance - amount;
}
_balances[recipient] += amount;

emit Transfer(sender, recipient, amount);

_afterTokenTransfer(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:
 *
 * - `account` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: 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 Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements:
```

```

*
* - `account` cannot be the zero address.
* - `account` must have at least `amount` tokens.
*/
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

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

    uint256 accountBalance = _balances[account];
    require(accountBalance >= amount, "ERC20: 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), "ERC20: approve from the zero address");
    //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
    require(spender != address(0), "ERC20: approve to the zero address");

```

```

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

/**
 * @dev Hook that is called before any transfer of tokens. This includes
 * minting and burning.
 *
 * Calling conditions:
 *
 * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
 * will be 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 {}
}

```

ERC20Burnable.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/ERC20Burnable.sol)

pragma solidity ^0.8.0;

import "./ERC20.sol";
import "./Context.sol";

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    //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 virtual {
    uint256 currentAllowance = allowance(account, _msgSender());
    require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
    unchecked {
        _approve(account, _msgSender(), currentAllowance - amount);
    }
    _burn(account, amount);
}
}
```

ERC20Pausable.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/ERC20Pausable.sol)

pragma solidity ^0.8.0;

import "./ERC20.sol";
import "./Pausable.sol";

/**
 * @dev ERC20 token with pausable token transfers, minting and burning.
 *
 * Useful for scenarios such as preventing trades until the end of an evaluation
 * period, or having an emergency switch for freezing all token transfers in the
 * event of a large bug.
 */
abstract contract ERC20Pausable is ERC20, Pausable {
    /**
     * @dev See {ERC20-_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(), "ERC20Pausable: token transfer while paused");
    }
}

```

AccessControlEnumerable.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (access/AccessControlEnumerable.sol)

pragma solidity ^0.8.0;

import "./IAccessControlEnumerable.sol";
import "./AccessControl.sol";
import "./EnumerableSet.sol";

/**
 * @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 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 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);
}
}
```

Context.sol

```
// SPDX-License-Identifier: MIT
// 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;
    }
}

```

IERC20.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/IERC20.sol)

pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);
}

```

```
/**
 * @dev Moves `amount` tokens from the caller's account to `recipient`.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transfer(address recipient, uint256 amount) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns
(uint256);

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an {Approval} event.
 */
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `sender` to `recipient` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
```

```

function transferFrom(
    address sender,
    address recipient,
    uint256 amount
) external returns (bool);

/**
 * @dev Emitted when `value` tokens are moved from one account (`from`) to
 * another (`to`).
 *
 * Note that `value` may be zero.
 */
event Transfer(address indexed from, address indexed to, uint256 value);

/**
 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
 * a call to {approve}. `value` is the new allowance.
 */
event Approval(address indexed owner, address indexed spender, uint256 value);
}
    
```

IERC20Metadata.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/IERC20Metadata.sol)

pragma solidity ^0.8.0;

import "./IERC20.sol";

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     *
     */
}
    
```

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

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

Pausable.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (security/Pausable.sol)

pragma solidity ^0.8.0;

import "./Context.sol";

/**
 * @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());
}
}

```

IAccessControlEnumerable.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (access/IAccessControlEnumerable.sol)

pragma solidity ^0.8.0;

import "./IAccessControl.sol";

/**
 * @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);
}

```

AccessControl.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (access/AccessControl.sol)

pragma solidity ^0.8.0;

import "./IAccessControl.sol";
import "./Context.sol";
import "./Strings.sol";
import "./ERC165.sol";

/**
 * @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, _msgSender());
    _;
}

/**
 * @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 override returns
(bool) {
    return _roles[role].members[account];
}

/**
 * @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 {
    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 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.
*/
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.
*/
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`.
*/
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.
 *
 * [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.
     */
    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.
     */
    function _revokeRole(bytes32 role, address account) internal virtual {
        if (hasRole(role, account)) {
            _roles[role].members[account] = false;
            emit RoleRevoked(role, account, _msgSender());
        }
    }
}
```

EnumerableSet.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/structs/EnumerableSet.sol)

pragma solidity ^0.8.0;

/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set\_\(abstract\_data\_type\) 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.
*/
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;
}
}

```

IAccessControl.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (access/IAccessControl.sol)

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;
}
    
```

Strings.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Strings.sol)

pragma solidity ^0.8.0;

/**
 * @dev String operations.
 */
library Strings {
    
```

```

bytes16 private constant _HEX_SYMBOLS = "0123456789abcdef";

/**
 * @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);
}
}

```

ERC165.sol

```

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/introspection/ERC165.sol)

pragma solidity ^0.8.0;

import "./IERC165.sol";

/**
 * @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;
    }
}

```

IERC165.sol

```

// SPDX-License-Identifier: MIT
// 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);
}

```

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](https://twitter.com/SlowMist_Team)



Github
<https://github.com/slowmist>