



Armors Labs

CoPuppy-NFT

Smart Contract Audit

- CoPuppy-NFT Audit Summary
- CoPuppy-NFT Audit
 - Document information
 - Audit results
 - Audited target file
 - Vulnerability analysis
 - Vulnerability distribution
 - Summary of audit results
 - Contract file
 - Analysis of audit results
 - Re-Entrancy
 - Arithmetic Over/Under Flows
 - Unexpected Blockchain Currency
 - Delegatecall
 - Default Visibilities
 - Entropy Illusion
 - External Contract Referencing
 - Unsolved TODO comments
 - Short Address/Parameter Attack
 - Unchecked CALL Return Values
 - Race Conditions / Front Running
 - Denial Of Service (DOS)
 - Block Timestamp Manipulation
 - Constructors with Care
 - Unintialised Storage Pointers
 - Floating Points and Numerical Precision
 - tx.origin Authentication
 - Permission restrictions

CoPuppy-NFT Audit Summary

Project name : CoPuppy-NFT Contract

Project address: None

Code URL : <https://github.com/copuppy/CP-NFT>

Commit : 0dc1bc7b3f2b336f7051b250fee4ee2dfe1c34a5

Project target : CoPuppy-NFT Contract Audit

Blockchain : Binance Smart Chain (BSC)

Test result : PASSED

Audit Info

Audit NO : 0X202107050007

Audit Team : Armors Labs

Audit Proofreading: <https://armors.io/#project-cases>

CoPuppy-NFT Audit

The CoPuppy-NFT team asked us to review and audit their CoPuppy-NFT contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
CoPuppy-NFT Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-07-05

Audit results

Note that: This contract is upgradable and this audit is only valid for the current version.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the CoPuppy-NFT contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report ("information provided" for short). Armors Labs postulates that the

information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
./UpgradeProxy.sol	b55feca9e511e0fa819bc3ab906d2a67
./CP-NFT-0624.sol	bde6ae11cef438fe74c09b64e3f112d7

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Uninitialised Storage Pointers	safe

Vulnerability	status
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
// SPDX-License-Identifier: UNLICENSED

// File: contracts/Utils/EnumerableSet.sol
pragma solidity >=0.6.0 <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
 *
 * 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
     * 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, th
     * 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 wit
            // 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;

            // When the value to delete is the last one, the swap operation is unnecessary. However,
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' stateme

            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] = toDeleteIndex + 1;
            // All indexes are 1-based

            // 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
    *
    * Note that there are no guarantees on the ordering of values
    * array, and it may change when more values are added
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
    }

    // 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
*
* Note that there are no guarantees on the ordering of v
* array, and it may change when more values are adde
*
* Requirements:
*
* - `index` must be strictly less than {length}.
*/
function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
    return _at(set._inner, index);
}

// 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 i
* 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, th
* 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
 *
 * Note that there are no guarantees on the ordering of v
 * array, and it may change when more values are added
 *
 * 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))));
}

// 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 i
 * 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, th
 * 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
     *
     * Note that there are no guarantees on the ordering of values
     * array, and it may change when more values are added.
     *
     * 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));
    }
}

// File: contracts/Utils/Address.sol

pragma solidity >=0.6.2 <0.8.0;

/**
 * @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
     *
     * 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 live
     * - an address where a contract lived, but
     *
     * =====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.
    }

```

```

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

/**
 * @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-che
 */
function sendValue(address payable recipient, uint256 amount) internal {
    require(address(this).balance >= amount, "Address: insufficient balance");

    // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
    (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
 * plain `call` is an unsafe replacement for a function call:
 * 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
 * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
 *
 * 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 (bool) {
    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 (bool) {
    return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
}

/**
* @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[functionCallWithValue],
* 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 (bool) {
    require(address(this).balance >= value, "Address: insufficient balance for call");
    require(isContract(target), "Address: call to non-contract");

    // solhint-disable-next-line avoid-low-level-calls
    (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");
}

```

```

// solhint-disable-next-line avoid-low-level-calls
(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
    require(isContract(target), "Address: delegate call to non-contract");

// solhint-disable-next-line avoid-low-level-calls
(bool success, bytes memory returndata) = target.delegatecall(data);
return _verifyCallResult(success, returndata, errorMessage);
}

function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) private
    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

            // solhint-disable-next-line no-inline-assembly
            assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
        } else {
            revert(errorMessage);
        }
    }
}

// File: contracts/utils/Context.sol

pragma solidity >=0.6.0 <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 GSN 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 payable) {
        return msg.sender;
    }

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

// File: contracts/access/AccessControl.sol

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Contract module that allows children to implement role-based access
 * control mechanisms.
 *
 * Roles are referred to by their `bytes32` identifier. These should
 * 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
 * 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 {
    using EnumerableSet for EnumerableSet.AddressSet;
    using Address for address;

    struct RoleData {
        EnumerableSet.AddressSet members;
        bytes32 adminRole;
    }

    mapping(bytes32 => RoleData) private _roles;

    bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

    /**
     * @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,
     * `bearer` except when using {_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) public view returns (bool) {
        return _roles[role].members.contains(account);
    }

    /**
     * @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 returns (uint256) {

```



```

        return _roles[role].members.length();
    }

    /**
     * @dev Returns one of the accounts that have `role`. `index` must be
     * 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
     * https://forum.openzeppelin.com/t/iterating-over-elements-on-enumerableset-in-openzeppelin-contracts/2296 for
     * more information.
     */
    function getRoleMember(bytes32 role, uint256 index) public view returns (address) {
        return _roles[role].members.at(index);
    }

    /**
     * @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 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 {
        require(hasRole(_roles[role].adminRole, _msgSender()), "AccessControl: sender must be an admin");
        _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 {
    require(hasRole(_roles[role].adminRole, _msgSender()), "AccessControl: sender must be an admin");

    _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
 *
 * If the calling account had been granted `role`, emits a
 * event.
 *
 * Requirements:
 *
 * - the caller must be `account`.
 */
function renounceRole(bytes32 role, address account) public virtual {
    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 construct
 * up the initial roles for the system.
 *
 * Using this function in any other way is effectively circumventing the admin
 * system imposed by {AccessControl}.
 * ====
 */
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 {
    emit RoleAdminChanged(role, _roles[role].adminRole, adminRole);
    _roles[role].adminRole = adminRole;
}

```

```

function _grantRole(bytes32 role, address account) private {
    if (_roles[role].members.add(account)) {
        emit RoleGranted(role, account, _msgSender());
    }
}

function _revokeRole(bytes32 role, address account) private {
    if (_roles[role].members.remove(account)) {
        emit RoleRevoked(role, account, _msgSender());
    }
}
}

```

// File: contracts/math/SafeMath.sol

```
pragma solidity >=0.6.0 <0.8.0;
```

```

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, with an
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an
     *
     *

```

```

* __Available since v3.4.__
*/
function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
    if (a == 0) return (true, 0);
    uint256 c = a * b;
    if (c / a != b) return (false, 0);
    return (true, c);
}

/**
 * @dev Returns the division of two unsigned integers, with
 *
 * __Available since v3.4.__
 */
function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (b == 0) return (false, 0);
    return (true, a / b);
}

/**
 * @dev Returns the remainder of dividing two unsigned integers, with
 *
 * __Available since v3.4.__
 */
function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (b == 0) return (false, 0);
    return (true, a % b);
}

/**
 * @dev Returns the addition of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `+` operator.
 *
 * Requirements:
 *
 * - Addition cannot overflow.
 */
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");
    return c;
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:

```

```

* - Subtraction cannot overflow.
*/
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a, "SafeMath: subtraction overflow");
    return a - b;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `*` operator.
 *
 * Requirements:
 *
 * - Multiplication cannot overflow.
 */
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) return 0;
    uint256 c = a * b;
    require(c / a == b, "SafeMath: multiplication overflow");
    return c;
}

/**
 * @dev Returns the integer division of two unsigned integers, reverting on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses
 * a `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b > 0, "SafeMath: division by zero");
    return a / b;
}

/**
 * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
 * reverting when dividing by zero.
 *
 * Counterpart to Solidity's `%` operator. This function uses
 * a `revert` opcode (which leaves remaining gas untouched) while Solidity uses
 * an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b > 0, "SafeMath: modulo by zero");

```

```

    return a % b;
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting with custom message
 * if overflow (when the result is negative).
 *
 * CAUTION: This function is deprecated because it requires allocating memory for the
 * message unnecessarily. For custom revert reasons use {trySub}.
 *
 * Counterpart to Solidity's '-' operator.
 *
 * Requirements:
 *
 * - Subtraction cannot overflow.
 */
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b <= a, errorMessage);
    return a - b;
}

/**
 * @dev Returns the integer division of two unsigned integers, reverting with custom
 * message if division by zero. The result is rounded towards zero.
 *
 * CAUTION: This function is deprecated because it requires allocating memory for the
 * message unnecessarily. For custom revert reasons use {tryDiv}.
 *
 * Counterpart to Solidity's '/' operator. Note: this function uses a
 * 'revert' opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b > 0, errorMessage);
    return a / b;
}

/**
 * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
 * arithmetic, i.e. no sign bits)
 *
 * CAUTION: This function is deprecated because it requires allocating memory for the
 * message unnecessarily. For custom revert reasons use {tryMod}.
 *
 * Counterpart to Solidity's '%' operator. This function uses a 'revert'
 * opcode (which leaves remaining gas untouched) while Solidity uses an
 * invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:

```

```

*
* - The divisor cannot be zero.
*/
function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b > 0, errorMessage);
    return a % b;
}

// File: contracts/utils/Counters.sol

pragma solidity >=0.6.0 <0.8.0;

/**
 * @title Counters
 * @author Matt Condon (@shrugs)
 * @dev Provides counters that can only be incremented or decremented by one. This can be used e.g. to track
 * of elements in a mapping, issuing ERC721 ids, or counting request ids.
 *
 * Include with `using Counters for Counters.Counter;`
 * Since it is not possible to overflow a 256 bit integer with increments of one, `increment`
 * overflow check, thereby saving gas. This does assume however correct usage, in that the
 * directly accessed.
 */
library Counters {
    using SafeMath for uint256;

    struct Counter {
        // This variable should never be directly accessed by users of the library: interactions must
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a
        // this feature: see https://github.com/ethereum/solidity/issues/4637
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        // The {SafeMath} overflow check can be skipped here, see the comment at the top
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}

// File: contracts/introspection/IERC165.sol

pragma solidity >=0.6.0 <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-
     * 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/token/ERC721/IERC721.sol

pragma solidity >=0.6.2 <0.8.0;

    /**
     * @dev Required interface of an ERC721 compliant contract.
     */
    interface IERC721 is IERC165 {
        /**
         * @dev Emitted when `tokenId` token is transferred from `from` to `to`.
         */
        event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);

        /**
         * @dev Emitted when `owner` enables `approved` to manage the `tokenId` token.
         */
        event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);

        /**
         * @dev Emitted when `owner` enables or disables (`approved`) `operator` to manage all of its assets.
         */
        event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

        /**
         * @dev Returns the number of tokens in `owner`'s account.
         */
        function balanceOf(address owner) external view returns (uint256 balance);

        /**
         * @dev Returns the owner of the `tokenId` token.
         *
         * Requirements:
         *
         * - `tokenId` must exist.

```

```

*/
    function ownerOf(uint256 tokenId) external view returns (address owner);

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients
     *         are aware of the ERC721 protocol to prevent tokens from being lost.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If the caller is not `from`, it must be have been allowed to move this token by either {approve} or {transferFrom}.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(address from, address to, uint256 tokenId) external;

    /**
     * @dev Transfers `tokenId` token from `from` to `to`.
     *
     * WARNING: Usage of this method is discouraged, use {safeTransferFrom} whenever possible.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must be owned by `from`.
     * - If the caller is not `from`, it must be approved to move this token by either {approve} or {transferFrom}.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address from, address to, uint256 tokenId) external;

    /**
     * @dev Gives permission to `to` to transfer `tokenId` token to another account.
     * The approval is cleared when the token is transferred.
     *
     * Only a single account can be approved at a time,
     *
     * Requirements:
     *
     * - The caller must own the token or be an approved operator.
     * - `tokenId` must exist.
     *
     * Emits an {Approval} event.
     */
    function approve(address to, uint256 tokenId) external;

    /**
     * @dev Returns the account approved for `tokenId` token.

```

```

*
* Requirements:
*
* - `tokenId` must exist.
*/
function getApproved(uint256 tokenId) external view returns (address operator);

/**
 * @dev Approve or remove `operator` as an operator for the
 * Operators can call {transferFrom} or {safeTransferFrom} for any token owned by the
 *
 * Requirements:
 *
 * - The `operator` cannot be the caller.
 *
 * Emits an {ApprovalForAll} event.
 */
function setApprovalForAll(address operator, bool _approved) external;

/**
 * @dev Returns if the `operator` is allowed to manage all of the
 *
 * See {setApprovalForAll}
 */
function isApprovedForAll(address owner, address operator) external view returns (bool);

/**
 * @dev Safely transfers `tokenId` token from `from` to `to`.
 *
 * Requirements:
 *
 * - `from` cannot be the zero address.
 * - `to` cannot be the zero address.
 * - `tokenId` token must exist and be owned by `from`.
 * - If the caller is not `from`, it must be approved to move this token by either {approve} or {setApprovalForAll}.
 * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}
 *
 * Emits a {Transfer} event.
 */
function safeTransferFrom(address from, address to, uint256 tokenId, bytes calldata data) external
}

// File: contracts/token/ERC721/IERC721Metadata.sol

pragma solidity >=0.6.2 <0.8.0;

/**
 * @title ERC-721 Non-Fungible Token Standard, optional metadata extension
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
interface IERC721Metadata is IERC721 {

```

```

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

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

        /**
        * @dev Returns the Uniform Resource Identifier (URI) for `tokenId` token.
        */
        function tokenId(uint256 tokenId) external view returns (string memory);
    }

    // File: contracts/token/ERC721/IERC721Enumerable.sol

    pragma solidity >=0.6.2 <0.8.0;

    /**
    * @title ERC-721 Non-Fungible Token Standard, optional enumeration extension
    * @dev See https://eips.ethereum.org/EIPS/eip-721
    */
    interface IERC721Enumerable is IERC721 {

        /**
        * @dev Returns the total amount of tokens stored by the
        */
        function totalSupply() external view returns (uint256);

        /**
        * @dev Returns a token ID owned by `owner` at a given
        * Use along with {balanceOf} to enumerate all of ``owner``'s tokens.
        */
        function tokenOfOwnerByIndex(address owner, uint256 index) external view returns (uint256 tokenId);

        /**
        * @dev Returns a token ID at a given `index` of all
        * Use along with {totalSupply} to enumerate all tokens.
        */
        function tokenByIndex(uint256 index) external view returns (uint256);
    }

    // File: contracts/token/ERC721/IERC721Receiver.sol

    pragma solidity >=0.6.0 <0.8.0;

    /**
    * @title ERC721 token receiver interface
    * @dev Interface for any contract that wants to support safeTransfers

```

```

* from ERC721 asset contracts.
*/
interface IERC721Receiver {
    /**
     * @dev Whenever an {IERC721} `tokenId` token is transferred to this contract via {IERC721Receiver:transferFrom}
     * by `operator` from `from`, this function is called.
     *
     * It must return its Solidity selector to confirm the token transfer.
     * If any other value is returned or the interface is not implemented by
     *
     * The selector can be obtained in Solidity with `IERC721.onERC721Received.selector`.
     */
    function onERC721Received(address operator, address from, uint256 tokenId, bytes calldata data) external returns (bytes4)
}

// File: contracts/introspection/ERC165.sol

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts may inherit from this and call {_registerInterface} to declare
 * their support of an interface.
 */
abstract contract ERC165 is IERC165 {
    /**
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;

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

    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of
     * `interfaceId`. Support of the actual ERC165 interface is automatic and

```

```

* registering its interface id is not required.
*
* See {IERC165-supportsInterface}.
*
* Requirements:
*
* - `interfaceId` cannot be the ERC165 invalid interface (0xffffffff).
*/
function _registerInterface(bytes4 interfaceId) internal virtual {
    require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
    _supportedInterfaces[interfaceId] = true;
}

// File: contracts/utils/EnumerableMap.sol

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Library for managing an enumerable variant of Solidity's
 * https://solidity.readthedocs.io/en/latest/types.html#mapping-types["mapping"]
 * type.
 *
 * Maps have the following properties:
 *
 * - Entries are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Entries are enumerated in O(n). No guarantees are made
 *
 * contract Example {
 *     // Add the library methods
 *     using EnumerableMap for EnumerableMap.UintToAddressMap;
 *
 *     // Declare a set state variable
 *     EnumerableMap.UintToAddressMap private myMap;
 * }
 *
 * As of v3.0.0, only maps of type `uint256 -> address` (UintToAddressMap) are
 * supported.
 */
library EnumerableMap {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Map type with
    // bytes32 keys and values.
    // The Map implementation uses private functions, and user-facing
    // implementations (such as Uint256ToAddressMap) are just wrappers around
    // the underlying Map.
    // This means that we can only create new EnumerableMaps for types that fit
    // in bytes32.

    struct MapEntry {
        bytes32 _key;
    }

```

```

    bytes32 _value;
}

struct Map {
    // Storage of map keys and values
    MapEntry[] _entries;

    // Position of the entry defined by a key in the `entries` array, plus 1
    // because index 0 means a key is not in the map.
    mapping(bytes32 => uint256) _indexes;
}

/**
 * @dev Adds a key-value pair to a map, or updates
 * key. O(1).
 *
 * Returns true if the key was added to the map, that is if
 * already present.
 */
function _set(Map storage map, bytes32 key, bytes32 value) private returns (bool) {
    // We read and store the key's index to prevent multiple reads from the same storage slot
    uint256 keyIndex = map._indexes[key];

    if (keyIndex == 0) { // Equivalent to !contains(map, key)
        map._entries.push(MapEntry({_key : key, _value : value}));
        // The entry is stored at length-1, but we add 1 to all indexes
        // and use 0 as a sentinel value
        map._indexes[key] = map._entries.length;
        return true;
    } else {
        map._entries[keyIndex - 1]._value = value;
        return false;
    }
}

/**
 * @dev Removes a key-value pair from a map. O(1).
 *
 * Returns true if the key was removed from the map, the
 */
function _remove(Map storage map, bytes32 key) private returns (bool) {
    // We read and store the key's index to prevent multiple reads from the same storage slot
    uint256 keyIndex = map._indexes[key];

    if (keyIndex != 0) { // Equivalent to contains(map, key)
        // To delete a key-value pair from the _entries array in O(1), we swap the entry to delete
        // in the array, and then remove the last entry (sometimes called as 'swap and pop').
        // This modifies the order of the array, as noted in {at}.

        uint256 toDeleteIndex = keyIndex - 1;
        uint256 lastIndex = map._entries.length - 1;

        // When the entry to delete is the last one, the swap operation is unnecessary. However,
        // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' statement

        MapEntry storage lastEntry = map._entries[lastIndex];

        // Move the last entry to the index where the entry to delete is
        map._entries[toDeleteIndex] = lastEntry;
        // Update the index for the moved entry
        map._indexes[lastEntry._key] = toDeleteIndex + 1; // All indexes are 1-based
    }
}

```



```

        // Delete the slot where the moved entry was stored
        map._entries.pop();

        // Delete the index for the deleted slot
        delete map._indexes[key];

        return true;
    } else {
        return false;
    }
}

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

/**
 * @dev Returns the number of key-value pairs in the m
 */
function _length(Map storage map) private view returns (uint256) {
    return map._entries.length;
}

/**
 * @dev Returns the key-value pair stored at position `index` in the
 *
 * Note that there are no guarantees on the ordering of e
 * array, and it may change when more entries are addec
 *
 * Requirements:
 *
 * - `index` must be strictly less than {length}.
 */
function _at(Map storage map, uint256 index) private view returns (bytes32, bytes32) {
    require(map._entries.length > index, "EnumerableMap: index out of bounds");

    MapEntry storage entry = map._entries[index];
    return (entry._key, entry._value);
}

/**
 * @dev Tries to returns the value associated with `key`. O(1).
 * Does not revert if `key` is not in the map.
 */
function _tryGet(Map storage map, bytes32 key) private view returns (bool, bytes32) {
    uint256 keyIndex = map._indexes[key];
    if (keyIndex == 0) return (false, 0); // Equivalent to contains(map, key)
    return (true, map._entries[keyIndex - 1]._value); // All indexes are 1-based
}

/**
 * @dev Returns the value associated with `key`. O(1).
 *
 * Requirements:
 *

```

```

*- `key` must be in the map.
*/
function _get(Map storage map, bytes32 key) private view returns (bytes32) {
    uint256 keyIndex = map._indexes[key];
    require(keyIndex != 0, "EnumerableMap: nonexistent key"); // Equivalent to contains(map, key)
    return map._entries[keyIndex - 1]._value; // All indexes are 1-based
}

/**
* @dev Same as [_get], with a custom error message when `key` is not in
*
* CAUTION: This function is deprecated because it requires allocating memory for the
* message unnecessarily. For custom revert reasons use [_tryGet].
*/
function _get(Map storage map, bytes32 key, string memory errorMessage) private view returns (bytes32) {
    uint256 keyIndex = map._indexes[key];
    require(keyIndex != 0, errorMessage); // Equivalent to contains(map, key)
    return map._entries[keyIndex - 1]._value; // All indexes are 1-based
}

// UintToAddressMap

struct UintToAddressMap {
    Map _inner;
}

/**
* @dev Adds a key-value pair to a map, or updates
* key. O(1).
*
* Returns true if the key was added to the map, that is if
* already present.
*/
function set(UintToAddressMap storage map, uint256 key, address value) internal returns (bool) {
    return _set(map._inner, bytes32(key), bytes32(uint256(uint160(value))));
}

/**
* @dev Removes a value from a set. O(1).
*
* Returns true if the key was removed from the map, the
*/
function remove(UintToAddressMap storage map, uint256 key) internal returns (bool) {
    return _remove(map._inner, bytes32(key));
}

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

/**
* @dev Returns the number of elements in the map. O(
*/
function length(UintToAddressMap storage map) internal view returns (uint256) {
    return _length(map._inner);
}

```

```

    }

    /**
     * @dev Returns the element stored at position `index` in the
     * Note that there are no guarantees on the ordering of v
     * array, and it may change when more values are added
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintToAddressMap storage map, uint256 index) internal view returns (uint256, address)
    {
        (bytes32 key, bytes32 value) = _at(map._inner, index);
        return (uint256(key), address(uint160(uint256(value))));
    }

    /**
     * @dev Tries to returns the value associated with `key`. O(1).
     * Does not revert if `key` is not in the map.
     *
     * __Available since v3.4.__
     */
    function tryGet(UintToAddressMap storage map, uint256 key) internal view returns (bool, address)
    {
        (bool success, bytes32 value) = _tryGet(map._inner, bytes32(key));
        return (success, address(uint160(uint256(value))));
    }

    /**
     * @dev Returns the value associated with `key`. O(1).
     *
     * Requirements:
     *
     * - `key` must be in the map.
     */
    function get(UintToAddressMap storage map, uint256 key) internal view returns (address) {
        return address(uint160(uint256(_get(map._inner, bytes32(key)))));
    }

    /**
     * @dev Same as {get}, with a custom error message when `key` is not in
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the
     * message unnecessarily. For custom revert reasons use {tryGet}.
     */
    function get(UintToAddressMap storage map, uint256 key, string memory errorMessage) internal view
    {
        return address(uint160(uint256(_get(map._inner, bytes32(key), errorMessage))));
    }
}

// File: contracts/Utils/Strings.sol

pragma solidity >=0.6.0 <0.8.0;

/**

```

```

* @dev String operations.
*/
library Strings {
    /**
    * @dev Converts a uint256 to its ASCII string 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/ora

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

// File: contracts/token/ERC721/ERC721.sol

pragma solidity >=0.6.0 <0.8.0;

    /**
    * @title ERC721 Non-Fungible Token Standard basic implementation
    * @dev see https://eips.ethereum.org/EIPS/eip-721
    */
    contract ERC721 is Context, ERC165, IERC721, IERC721Metadata, IERC721Enumerable {
        using SafeMath for uint256;
        using Address for address;
        using EnumerableSet for EnumerableSet.UintSet;
        using EnumerableMap for EnumerableMap.UintToAddressMap;
        using Strings for uint256;

        // Equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
        // which can be also obtained as `IERC721Receiver(0).onERC721Received.selector`
        bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;

        // Mapping from holder address to their (enumerable) set of owned tokens

```

```

mapping(address => EnumerableSet.UintSet) private _holderTokens;

// Enumerable mapping from token ids to their owners
EnumerableMap.UintToAddressMap private _tokenOwners;

// Mapping from token ID to approved address
mapping(uint256 => address) private _tokenApprovals;

// Mapping from owner to operator approvals
mapping(address => mapping(address => bool)) private _operatorApprovals;

// Token name
string private _name;

// Token symbol
string private _symbol;

// Optional mapping for token URIs
mapping(uint256 => string) private _tokenURIs;

// Base URI
string private _baseURI;

/*
 * bytes4(keccak256('balanceOf(address)')) == 0x70a08231
 * bytes4(keccak256('ownerOf(uint256)')) == 0x6352211e
 * bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
 * bytes4(keccak256('getApproved(uint256)')) == 0x081812fc
 * bytes4(keccak256('setApprovalForAll(address,bool)')) == 0xa22cb465
 * bytes4(keccak256('isApprovedForAll(address,address)')) == 0xe985e9c5
 * bytes4(keccak256('transferFrom(address,address,uint256)')) == 0x23b872dd
 * bytes4(keccak256('safeTransferFrom(address,address,uint256)')) == 0x42842e0e
 * bytes4(keccak256('safeTransferFrom(address,address,uint256,bytes)')) == 0xb88d4fde
 *
 * ==> 0x70a08231 ^ 0x6352211e ^ 0x095ea7b3 ^ 0x081812fc ^
 *      0xa22cb465 ^ 0xe985e9c5 ^ 0x23b872dd ^ 0x42842e0e ^ 0xb88d4fde == 0x80ac58cd
 */
bytes4 private constant _INTERFACE_ID_ERC721 = 0x80ac58cd;

/*
 * bytes4(keccak256('name()')) == 0x06fdde03
 * bytes4(keccak256('symbol()')) == 0x95d89b41
 * bytes4(keccak256('tokenURI(uint256)')) == 0xc87b56dd
 *
 * ==> 0x06fdde03 ^ 0x95d89b41 ^ 0xc87b56dd == 0x5b5e139f
 */
bytes4 private constant _INTERFACE_ID_ERC721_METADATA = 0x5b5e139f;

/*
 * bytes4(keccak256('totalSupply()')) == 0x18160ddd
 * bytes4(keccak256('tokenOfOwnerByIndex(address,uint256)')) == 0x2f745c59
 * bytes4(keccak256('tokenByIndex(uint256)')) == 0x4f6ccce7
 *
 * ==> 0x18160ddd ^ 0x2f745c59 ^ 0x4f6ccce7 == 0x780e9d63
 */
bytes4 private constant _INTERFACE_ID_ERC721_ENUMERABLE = 0x780e9d63;

/**
 * @dev Initializes the contract by setting a `name` and
 */
function initializeA(string memory name_, string memory symbol_) internal {
    _name = name_;
    _symbol = symbol_;

    // register the supported interfaces to conform to ERC721 via ERC165

```

```

    _registerInterface(_INTERFACE_ID_ERC721);
    _registerInterface(_INTERFACE_ID_ERC721_METADATA);
    _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
}

/**
 * @dev See {IERC721-balanceOf}.
 */
function balanceOf(address owner) public view virtual override returns (uint256) {
    require(owner != address(0), "ERC721: balance query for the zero address");
    return _holderTokens[owner].length();
}

/**
 * @dev See {IERC721-ownerOf}.
 */
function ownerOf(uint256 tokenId) public view virtual override returns (address) {
    return _tokenOwners.get(tokenId, "ERC721: owner query for nonexistent token");
}

/**
 * @dev See {IERC721Metadata-name}.
 */
function name() public view virtual override returns (string memory) {
    return _name;
}

/**
 * @dev See {IERC721Metadata-symbol}.
 */
function symbol() public view virtual override returns (string memory) {
    return _symbol;
}

/**
 * @dev See {IERC721Metadata-tokenURI}.
 */
function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
    require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

    string memory _tokenURI = _tokenURIs[tokenId];
    string memory base = baseURI();

    // If there is no base URI, return the token URI.
    if (bytes(base).length == 0) {
        return _tokenURI;
    }
    // If both are set, concatenate the baseURI and tokenURI (via abi.encodePacked).
    if (bytes(_tokenURI).length > 0) {
        return string(abi.encodePacked(base, _tokenURI));
    }
    // If there is a baseURI but no tokenURI, concatenate the tokenId to the baseURI.
    return string(abi.encodePacked(base, tokenId.toString()));
}

/**
 * @dev Returns the base URI set via {_setBaseURI}. This will
 * automatically added as a prefix in {tokenURI} to each token's URI, or
 * to the token ID if no specific URI is set for that token ID.
 */

```

```

function baseURI() public view virtual returns (string memory) {
    return _baseURI;
}

/**
 * @dev See {IERC721Enumerable-tokenOfOwnerByIndex}.
 */
function tokenOfOwnerByIndex(address owner, uint256 index) public view virtual override returns (
    return _holderTokens[owner].at(index);
}

/**
 * @dev See {IERC721Enumerable-totalSupply}.
 */
function totalSupply() public view virtual override returns (uint256) {
    // _tokenOwners are indexed by tokenIds, so .length() returns the number of tokenIds
    return _tokenOwners.length();
}

/**
 * @dev See {IERC721Enumerable-tokenByIndex}.
 */
function tokenByIndex(uint256 index) public view virtual override returns (uint256) {
    (uint256 tokenId,) = _tokenOwners.at(index);
    return tokenId;
}

/**
 * @dev See {IERC721-approve}.
 */
function approve(address to, uint256 tokenId) public virtual override {
    address owner = ERC721.ownerOf(tokenId);
    require(to != owner, "ERC721: approval to current owner");

    require(_msgSender() == owner || ERC721.isApprovedForAll(owner, _msgSender()),
        "ERC721: approve caller is not owner nor approved for all"
    );

    _approve(to, tokenId);
}

/**
 * @dev See {IERC721-getApproved}.
 */
function getApproved(uint256 tokenId) public view virtual override returns (address) {
    require(_exists(tokenId), "ERC721: approved query for nonexistent token");

    return _tokenApprovals[tokenId];
}

/**
 * @dev See {IERC721-setApprovalForAll}.
 */
function setApprovalForAll(address operator, bool approved) public virtual override {
    require(operator != _msgSender(), "ERC721: approve to caller");

    _operatorApprovals[_msgSender()][operator] = approved;
    emit ApprovalForAll(_msgSender(), operator, approved);
}

```



```

/**
 * @dev See {IERC721-isApprovedForAll}.
 */
function isApprovedForAll(address owner, address operator) public view virtual override returns (
    return _operatorApprovals[owner][operator];
}

/**
 * @dev See {IERC721-transferFrom}.
 */
function transferFrom(address from, address to, uint256 tokenId) public virtual override {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor
    _transfer(from, to, tokenId);
}

/**
 * @dev See {IERC721-safeTransferFrom}.
 */
function safeTransferFrom(address from, address to, uint256 tokenId) public virtual override {
    safeTransferFrom(from, to, tokenId, "");
}

/**
 * @dev See {IERC721-safeTransferFrom}.
 */
function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data) public v
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor
    _safeTransfer(from, to, tokenId, _data);
}

/**
 * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients
 * are aware of the ERC721 protocol to prevent tokens from being lost.
 *
 * `_data` is additional data, it has no specified format and it is sent in call to `to`.
 *
 * This internal function is equivalent to {safeTransferFrom}, and can be used to e.g.
 * implement alternative mechanisms to perform token transfer, such as signature-based
 *
 * Requirements:
 *
 * - `from` cannot be the zero address.
 * - `to` cannot be the zero address.
 * - `tokenId` token must exist and be owned by `from`.
 * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}
 *
 * Emits a {Transfer} event.
 */
function _safeTransfer(address from, address to, uint256 tokenId, bytes memory _data) internal vi
    _transfer(from, to, tokenId);
    require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non ERC721Rece
}

/**

```

```

* @dev Returns whether `tokenId` exists.
*
* Tokens can be managed by their owner or approved accounts via {approve} or {setApprovalForAll}.
*
* Tokens start existing when they are minted (_mint),
* and stop existing when they are burned (_burn).
*/
function _exists(uint256 tokenId) internal view virtual returns (bool) {
    return _tokenOwners.contains(tokenId);
}

/**
* @dev Returns whether `spender` is allowed to manage `tokenId`.
*
* Requirements:
*
* - `tokenId` must exist.
*/
function _isApprovedOrOwner(address spender, uint256 tokenId) internal view virtual returns (bool) {
    require(_exists(tokenId), "ERC721: operator query for nonexistent token");
    address owner = ERC721.ownerOf(tokenId);
    return (spender == owner || getApproved(tokenId) == spender || ERC721.isApprovedForAll(owner,
}

/**
* @dev Safely mints `tokenId` and transfers it to `to`.
*
* Requirements:
*
* - `tokenId` must not exist.
* - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}.
*
* Emits a {Transfer} event.
*/
function _safeMint(address to, uint256 tokenId) internal virtual {
    _safeMint(to, tokenId, "");
}

/**
* @dev Same as {xref-ERC721-_safeMint-address-uint256-}[_safeMint], with an ac
* forwarded in {IERC721Receiver-onERC721Received} to contract recipients.
*/
function _safeMint(address to, uint256 tokenId, bytes memory _data) internal virtual {
    _mint(to, tokenId);
    require(_checkOnERC721Received(address(0), to, tokenId, _data), "ERC721: transfer to non ERC7
}

/**
* @dev Mints `tokenId` and transfers it to `to`.
*
* WARNING: Usage of this method is discouraged, use {_safeMint} whenever possible
*
* Requirements:
*
* - `tokenId` must not exist.

```

```

* - `to` cannot be the zero address.
*
* Emits a {Transfer} event.
*/
function _mint(address to, uint256 tokenId) internal virtual {
    require(to != address(0), "ERC721: mint to the zero address");
    require(!_exists(tokenId), "ERC721: token already minted");

    _beforeTokenTransfer(address(0), to, tokenId);

    _holderTokens[to].add(tokenId);

    _tokenOwners.set(tokenId, to);

    emit Transfer(address(0), to, tokenId);
}

/**
* @dev Destroys `tokenId`.
* The approval is cleared when the token is burned.
*
* Requirements:
*
* - `tokenId` must exist.
*
* Emits a {Transfer} event.
*/
function _burn(uint256 tokenId) internal virtual {
    address owner = ERC721.ownerOf(tokenId);
    // internal owner

    _beforeTokenTransfer(owner, address(0), tokenId);

    // Clear approvals
    _approve(address(0), tokenId);

    // Clear metadata (if any)
    if (bytes(_tokenURIs[tokenId]).length != 0) {
        delete _tokenURIs[tokenId];
    }

    _holderTokens[owner].remove(tokenId);

    _tokenOwners.remove(tokenId);

    emit Transfer(owner, address(0), tokenId);
}

/**
* @dev Transfers `tokenId` from `from` to `to`.
* As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
*
* Requirements:
*
* - `to` cannot be the zero address.
* - `tokenId` token must be owned by `from`.
*
* Emits a {Transfer} event.

```

```

*/
function _transfer(address from, address to, uint256 tokenId) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer of token that is not own");
    // internal owner
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    _approve(address(0), tokenId);

    _holderTokens[from].remove(tokenId);
    _holderTokens[to].add(tokenId);

    _tokenOwners.set(tokenId, to);

    emit Transfer(from, to, tokenId);
}

/**
 * @dev Sets `tokenId` as the tokenURI of `tokenId`.
 *
 * Requirements:
 *
 * - `tokenId` must exist.
 */
function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal virtual {
    require(!_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
    _tokenURIs[tokenId] = _tokenURI;
}

/**
 * @dev Internal function to set the base URI for all token IDs. It is
 * automatically added as a prefix to the value returned i
 * or to the token ID if {tokenURI} is empty.
 */
function _setBaseURI(string memory baseURI_) internal virtual {
    _baseURI = baseURI_;
}

/**
 * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a target
 * The call is not executed if the target address is not a contract
 *
 * @param from address representing the previous owner of the token
 * @param to target address that will receive the tokens
 * @param tokenId uint256 ID of the token to be transferred
 * @param _data bytes optional data to send along with the call
 * @return bool whether the call correctly returned the expected token
 */
function _checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory _data)
private returns (bool)
{
    if (!to.isContract()) {
        return true;
    }
    bytes memory returndata = to.functionCall(abi.encodeWithSelector(
        IERC721Receiver(to).onERC721Received.selector,
        _msgSender(),

```

```

        from,
        tokenId,
        _data
    ), "ERC721: transfer to non ERC721Receiver implementer");
    bytes4 retval = abi.decode(returndata, (bytes4));
    return (retval == _ERC721_RECEIVED);
}

function _approve(address to, uint256 tokenId) private {
    _tokenApprovals[tokenId] = to;
    emit Approval(ERC721.ownerOf(tokenId), to, tokenId);
    // internal owner
}

/**
 * @dev Hook that is called before any token transfer. This includes minting
 * and burning.
 *
 * Calling conditions:
 *
 * - When `from` and `to` are both non-zero, `from`s `tokenId` will
 *   transferred to `to`.
 * - When `from` is zero, `tokenId` will be minted for `to`.
 * - When `to` is zero, `from`s `tokenId` will be burned.
 * - `from` cannot be the zero address.
 * - `to` cannot be the zero address.
 *
 * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks
 */
function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal virtual {}
}

// File: contracts/token/ERC721/ERC721Burnable.sol

pragma solidity >=0.6.0 <0.8.0;

/**
 * @title ERC721 Burnable Token
 * @dev ERC721 Token that can be irreversibly burned (destroyed).
 */
abstract contract ERC721Burnable is Context, ERC721 {
    /**
     * @dev Burns `tokenId`. See {ERC721-burn}.
     *
     * Requirements:
     *
     * - The caller must own `tokenId` or be an approved operator.
     */
    function burn(uint256 tokenId) public virtual {
        //solhint-disable-next-line max-line-length
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721Burnable: caller is not owner nor a
        _burn(tokenId);
    }
}

```

```
// File: contracts/utils/Pausable.sol

pragma solidity >=0.6.0 <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 this simply including this module, only once the modifiers
 */
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 () internal {
        _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
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
        require(!paused(), "Pausable: paused");
        _;
    }

    /**
```

```

* @dev Modifier to make a function callable only when the
*
* 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/token/ERC721/ERC721Pausable.sol

pragma solidity >=0.6.0 <0.8.0;

/**
* @dev ERC721 token with pausable token transfers, minting and burning.
*
* Useful for scenarios such as preventing trades until the
* period, or having an emergency switch for freezing all token transfers in
* event of a large bug.
*/
abstract contract ERC721Pausable is ERC721, Pausable {
    /**
    * @dev See {ERC721-_beforeTokenTransfer}.
    *
    * Requirements:
    
```

```

*
* - the contract must not be paused.
*/
function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal virtual override
    super._beforeTokenTransfer(from, to, tokenId);

    require(!paused(), "ERC721Pausable: token transfer while paused");
}

}

// File: contracts/presets/ERC721PresetMinterPauserAutoId.sol

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev {ERC721} 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
 * - token ID and URI autogeneration
 *
 * This contract uses {AccessControl} to lock permissioned functions using
 * different roles - head to its documentation for details.
 *
 * The account that deploys the contract will be granted
 * roles, as well as the default admin role, which will let it
 * and pauser roles to other accounts.
 */
contract ERC721PresetMinterPauserAutoId is Context, AccessControl, ERC721Burnable, ERC721Pausable {
    using Counters for Counters.Counter;

    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");

    Counters.Counter private _tokenIdTracker;

    /**
     * @dev Grants `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE` and `PAUSER_ROLE` to
     * account that deploys the contract.
     *
     * Token URIs will be autogenerated based on `baseURI` and their token IDs.
     * See {ERC721-tokenURI}.
     */
    function initializeB(string memory name, string memory symbol, string memory baseURI) internal {
        initializeA(name, symbol);
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());

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

        _setBaseURI(baseURI);
    }
}

```



```

    /**
    * @dev Creates a new token for `to`. Its token ID will be
    * assigned (and available on the emitted {IERC721-Transfer} event), and
    * URI autogenerated based on the base URI passed at construction.
    *
    * See {ERC721-_mint}.
    *
    * Requirements:
    *
    * - the caller must have the `MINTER_ROLE`.
    */
    function mint(address to) public virtual returns (uint256){
        require(hasRole(MINTER_ROLE, _msgSender()), "ERC721PresetMinterPauserAutoId: must have minter role");

        // We cannot just use balanceOf to create the new tokenId because tokens
        // can be burned (destroyed), so we need a separate counter.
        uint256 id = _tokenIdTracker.current();
        _mint(to, id);
        _tokenIdTracker.increment();
        return id;
    }

    /**
    * @dev Pauses all token transfers.
    *
    * See {ERC721Pausable} and {Pausable-_pause}.
    *
    * Requirements:
    *
    * - the caller must have the `PAUSER_ROLE`.
    */
    function pause() public virtual {
        require(hasRole(PAUSER_ROLE, _msgSender()), "ERC721PresetMinterPauserAutoId: must have pauser role");
        _pause();
    }

    /**
    * @dev Unpauses all token transfers.
    *
    * See {ERC721Pausable} and {Pausable-_unpause}.
    *
    * Requirements:
    *
    * - the caller must have the `PAUSER_ROLE`.
    */
    function unpause() public virtual {
        require(hasRole(PAUSER_ROLE, _msgSender()), "ERC721PresetMinterPauserAutoId: must have pauser role");
        _unpause();
    }

    function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal virtual override {
        super._beforeTokenTransfer(from, to, tokenId);
    }

    function setTokenURI(uint256 tokenId, string memory _tokenURI) public virtual {
        require(hasRole(MINTER_ROLE, _msgSender()), "ERC721PresetMinterPauserAutoId: must have minter role");
        _setTokenURI(tokenId, _tokenURI);
    }

```

```

        _setTokenURI(tokenId, _tokenURI);
    }

    function setBaseURI(string memory baseURI_) public virtual {
        require(hasRole(MINTER_ROLE, _msgSender()), "ERC721PresetMinterPauserAutoId: must have minter");
        _setBaseURI(baseURI_);
    }
}

// File: contracts/CO-NFT.sol
interface ICPDelivery {
    function transferCall(address from,address to,uint256 _pid) external;
}

pragma solidity ^0.6.0;

contract CPNFT is ERC721PresetMinterPauserAutoId {
    //0x8129fc1c
    function initialize() public {
        require(msg.sender == 0x4e5CF0F9Ea9BB77B834958Faf3124c0C211302aB, 'not deploy');
        initializeB('CPNFT', 'CPNFT', '');
    }
    event ModifyPropEvent(uint256 tokenId);

    struct NFTProp {
        uint256 picId; // picId
        uint256 power; // power;
        uint256 gender; // gender
        uint256 level; // level
        uint256 weapon; // weapon
        uint256 lockedVal; // lockedVal
        uint256 lockedValType;
        uint256 saleVal;
        uint256 saleValType;
        uint256 sby10;
        uint256 sby11;
        uint256 sby12;
    }

    mapping(uint256 => NFTProp) public Props;

    function setNFTProps(uint256 tokenId, uint256[] memory _val) public {
        require(hasRole(MINTER_ROLE, _msgSender()), "ERC721PresetMinterPauserAutoId: must have minter");
        require(!_exists(tokenId), "ERC721Metadata: set of nonexistent token");
        Props[tokenId].picId = _val[0];
        Props[tokenId].power = _val[1];
        Props[tokenId].gender = _val[2];
        Props[tokenId].level = _val[3];
        Props[tokenId].weapon = _val[4];
        Props[tokenId].lockedVal = _val[5];
        emit ModifyPropEvent(tokenId);
    }

    function getLendType(uint256 tokenId) public view returns (uint256){
        return Props[tokenId].gender; //wait implement
    }

    function setNFTProp(uint256 tokenId, uint8 _index, uint256 _val) public virtual {
        require(hasRole(MINTER_ROLE, _msgSender()), "ERC721PresetMinterPauserAutoId: must have minter");
        require(!_exists(tokenId), "ERC721Metadata: set of nonexistent token");
        if (_index == 1) {
            Props[tokenId].picId = _val;
        }
        if (_index == 2) {
            Props[tokenId].power = _val;
        }
    }
}

```

```

    }
    if (_index == 3) {
        Props[tokenId].gender = _val;
    }
    if (_index == 4) {
        Props[tokenId].level = _val;
    }
    if (_index == 5) {
        Props[tokenId].weapon = _val;
    }
    if (_index == 6) {
        Props[tokenId].lockedVal = _val;
    }
    if (_index == 7) {
        Props[tokenId].lockedValType = _val;
    }
    if (_index == 8) {
        Props[tokenId].saleVal = _val;
    }
    if (_index == 9) {
        Props[tokenId].saleValType = _val;
    }
    if (_index == 10) {
        Props[tokenId].sby10 = _val;
    }
    if (_index == 11) {
        Props[tokenId].sby11 = _val;
    }
    if (_index == 12) {
        Props[tokenId].sby12 = _val;
    }
    emit ModifyPropEvent(tokenId);
}

function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal virtual override
    super._beforeTokenTransfer(from, to, tokenId);
    //logic of genesis card and shard card bonus
    //ICPDelivery().transferCall(from,to,0);
}

function transferList(address _sender, address _to, uint256[] calldata _tokenIdList) public {
    for (uint256 i = 0; i < _tokenIdList.length; i++) {
        safeTransferFrom(_sender, _to, _tokenIdList[i]);
    }
}

}

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

/**
 * @dev This abstract contract provides a fallback function that delegates all calls to an
 * instruction `delegatecall`. We refer to the second contract as the
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through
 * different contract through the {delegate} function.
 *
 * The success and return data of the delegated call will be
 */
abstract contract Proxy {

```

```

/**
 * @dev Delegates the current call to `implementation`.
 *
 * This function does not return to its internal call site, it will return directly to
 */
function _delegate(address implementation) internal {
    // solhint-disable-next-line no-inline-assembly
    assembly {
        // Copy msg.data. We take full control of memory in this inline assembly
        // block because it will not return to Solidity code. We overwrite the
        // Solidity scratch pad at memory position 0.
        calldatacopy(0, 0, calldatasize())

        // Call the implementation.
        // out and outsize are 0 because we don't know the size yet.
        let result := delegatecall(
            gas(),
            implementation,
            0,
            calldatasize(),
            0,
            0
        )

        // Copy the returned data.
        returndatacopy(0, 0, returndatasize())

        switch result
            // delegatecall returns 0 on error.
            case 0 {
                revert(0, returndatasize())
            }
            default {
                return(0, returndatasize())
            }
    }
}

/**
 * @dev This is a virtual function that should be override
 * and {_fallback} should delegate.
 */
function _implementation() internal view virtual returns (address);

/**
 * @dev Delegates the current call to the address return
 *
 * This function does not return to its internal call site, it will return directly to
 */
function _fallback() internal {
    // _beforeFallback();

    _delegate(_implementation());
}

/**
 * @dev Fallback function that delegates calls to the address returned by `_impleme
 * function in the contract matches the call data.
 */
fallback() external payable {

```

```

        _fallback();
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_implement
     * is empty.
     */
    receive() external payable {
        _fallback();
    }

    /**
     * @dev Hook that is called before falling back to the implementation. Can happen a
     * call, or as part of the Solidity `fallback` or `receive` functions.
     *
     * If overridden should call `super._beforeFallback()`.
     */
    function _beforeFallback() internal virtual {}
}

pragma solidity ^0.6.2;

/**
 * @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
     *
     * Among others, `isContract` will return false for the fol
     * types of addresses:
     *
     * - an externally-owned account
     * - a contract in construction
     * - an address where a contract will
     * - an address where a contract lived, but
     *
     * =====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
            0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            codehash := extcodehash(account)
        }
        return (codehash != accountHash && codehash != 0x0);
    }
}

```

```

/**
 * @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 the
 * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-che
 */
function sendValue(address payable recipient, uint256 amount) internal {
    require(
        address(this).balance >= amount,
        "Address: insufficient balance"
    );

    // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
    (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
 * plain `call` is an unsafe replacement for a function call:
 * 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
 * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
 *
 * 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
    * - 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],
    * 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"
        );
        return _functionCallWithValue(target, data, value, errorMessage);
    }

```

```

function _functionCallWithValue(
    address target,
    bytes memory data,
    uint256 weiValue,
    string memory errorMessage
) private returns (bytes memory) {
    require(isContract(target), "Address: call to non-contract");

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) =
        target.call{value: weiValue}(data);
    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

            // solhint-disable-next-line no-inline-assembly
            assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
        } else {
            revert(errorMessage);
        }
    }
}

pragma solidity ^0.6.0;

/**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because cal
 * implementation address that can be changed. This address is stored in storage in the
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally up
 * {TransparentUpgradeableProxy}.
 */
contract UpgradeableProxy is Proxy {
    /**
     * @dev Initializes the upgradeable proxy with an initial i
     *
     * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This
     * function call, and allows initializing the storage of the
     */
    constructor(address _logic, bytes memory _data) public payable {
        assert(
            _IMPLEMENTATION_SLOT ==
                bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1)
        );
        _setImplementation(_logic);
        if (_data.length > 0) {
            // solhint-disable-next-line avoid-low-level-calls
            (bool success, ) = _logic.delegatecall(_data);
            require(success);
        }
    }
}

```



```

    /**
     * @dev Emitted when the implementation is upgraded.
     */
    event Upgraded(address indexed implementation);

    /**
     * @dev Storage slot with the address of the current implementation.
     * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, and
     * validated in the constructor.
     */
    bytes32 private constant _IMPLEMENTATION_SLOT =
        0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;

    /**
     * @dev Returns the current implementation address.
     */
    function _implementation() internal view override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

    /**
     * @dev Upgrades the proxy to a new implementation.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Stores a new address in the EIP1967 implementation slot.
     */
    function _setImplementation(address newImplementation) private {
        require(
            Address.isContract(newImplementation),
            "UpgradeableProxy: new implementation is not a contract"
        );

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}

pragma solidity ^0.6.0;

/**
 * @dev This contract implements a proxy that is upgradeable by an
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357 [proxy self-destruct]

```

* clashing], which can potentially be used in an attack, this contract uses

* <https://blog.openzeppelin.com/the-transparent-proxy-pattern/> [transparent proxy pattern]

* things that go hand in hand:

*

* 1. If any account other than the admin calls the proxy,

* that call matches one of the admin functions exposed by the

* 2. If the admin calls the proxy, it can access

* implementation. If the admin tries to call a function on

* "admin cannot fallback to proxy target".

*

* These properties mean that the admin account can only be used for admin actions

* the admin, so it's best if it's a

* to sudden errors when trying to call a function from the p

*

* Our recommendation is for the dedicated account to be an

* you should think of the `Pro.

*/

```
contract TransparentUpgradeableProxy is UpgradeableProxy {
    /**
     * @dev Initializes an upgradeable proxy managed by `_admin`, backed by
     * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
     */
    constructor(
        address _logic,
        address _admin,
        bytes memory _data
    ) payable UpgradeableProxy(_logic, _data) {
        assert(
            _ADMIN_SLOT ==
            bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1)
        );
        _setAdmin(_admin);
    }

    /**
     * @dev Emitted when the admin account has changed.
     */
    event AdminChanged(address previousAdmin, address newAdmin);

    /**
     * @dev Storage slot with the admin of the contract.
     * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
     * validated in the constructor.
     */
    bytes32 private constant _ADMIN_SLOT =
        0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;

    /**
     * @dev Modifier used internally that will delegate the call
     */
    modifier ifAdmin() {
        if (msg.sender == _admin()) {
            _;
        } else {
            _fallback();
        }
    }
}
```

```

    }

    /**
     * @dev Returns the current admin.
     *
     * NOTE: Only the admin can call this function. See {Proxy}
     *
     * TIP: To get this value clients can read directly from the storage slot shown below (see
     * https://eth.wiki/json-rpc/API#eth\_getstorageat [eth_getStorageAt] RPC call.
     * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
     */
    function admin() external ifAdmin returns (address) {
        return _admin();
    }

    /**
     * @dev Returns the current implementation.
     *
     * NOTE: Only the admin can call this function. See {Proxy}
     *
     * TIP: To get this value clients can read directly from the storage slot shown below (see
     * https://eth.wiki/json-rpc/API#eth\_getstorageat [eth_getStorageAt] RPC call.
     * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
     */
    function implementation() external ifAdmin returns (address) {
        return _implementation();
    }

    /**
     * @dev Changes the admin of the proxy.
     *
     * Emits an {AdminChanged} event.
     *
     * NOTE: Only the admin can call this function. See {Proxy}
     */
    function changeAdmin(address newAdmin) external ifAdmin {
        require(
            newAdmin != address(0),
            "TransparentUpgradeableProxy: new admin is the zero address"
        );
        emit AdminChanged(_admin(), newAdmin);
        _setAdmin(newAdmin);
    }

    /**
     * @dev Upgrade the implementation of the proxy.
     *
     * NOTE: Only the admin can call this function. See {Proxy}
     */
    function upgradeTo(address newImplementation) external ifAdmin {
        _upgradeTo(newImplementation);
    }

    /**
     * @dev Upgrade the implementation of the proxy, and the
     * by `data`, which should be an encoded function call. T

```

```

*proxied contract.
*
*      NOTE:      Only      the      admin can call this function. See {Prox
*/
function upgradeToAndCall(address newImplementation, bytes calldata data)
    external
    payable
    ifAdmin
{
    _upgradeTo(newImplementation);
    // solhint-disable-next-line avoid-low-level-calls
    (bool success, ) = newImplementation.delegatecall(data);
    require(success);
}

/**
* @dev Returns      the      current admin.
*/
function _admin() internal view returns (address adm) {
    bytes32 slot = _ADMIN_SLOT;
    // solhint-disable-next-line no-inline-assembly
    assembly {
        adm := sload(slot)
    }
}

/**
* @dev Stores      a      new address in      the      EIP1967 admin slot
*/
function _setAdmin(address newAdmin) private {
    bytes32 slot = _ADMIN_SLOT;

    // solhint-disable-next-line no-inline-assembly
    assembly {
        sstore(slot, newAdmin)
    }
}

/**
* @dev Makes sure      the      admin cannot access      the      fallback
*/
function _beforeFallback() internal virtual override {
    require(
        msg.sender != _admin(),
        "TransparentUpgradeableProxy: admin cannot fallback to proxy target"
    );
    super._beforeFallback();
}
}

// File: contracts/IFOUUpgradeProxy.sol

pragma solidity 0.6.12;

contract UpgradeProxy is TransparentUpgradeableProxy {
    constructor(
        address admin,
        address logic,
        bytes memory data
    ) public TransparentUpgradeableProxy(logic, admin, data) {}
}

```

Analysis of audit results

Re-Entrancy

- **Description:**

One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Arithmetic Over/Under Flows

- **Description:**

The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Unexpected Blockchain Currency

- **Description:**

Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.

- **Detection results:**

PASSED!

- **Security suggestion:** no.

Delegatecall

- **Description:**

The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.

- **Detection results:**

PASSED!

- **Security suggestion:** no.

Default Visibilities

- **Description:**

Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devastating vulnerabilities in smart contracts as will be discussed in this section.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Entropy Illusion

- **Description:**

All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

External Contract Referencing

- **Description:**

One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general

operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Unsolved TODO comments

- **Description:**

Check for Unsolved TODO comments

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Short Address/Parameter Attack

- **Description:**

This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Unchecked CALL Return Values

- **Description:**

There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Race Conditions / Front Running

- **Description:**

The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Denial Of Service (DOS)

- **Description:**

This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Block Timestamp Manipulation

- **Description:**

Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Constructors with Care

- **Description:**

Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.

- **Detection results:**

PASSED!

- **Security suggestion:**
no.

Unintialised Storage Pointers

- **Description:**

The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately initialising variables.

- **Detection results:**

PASSED!

- **Security suggestion:**
no.

Floating Points and Numerical Precision

- **Description:**

As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.

- **Detection results:**

PASSED!

- **Security suggestion:**
no.

tx.origin Authentication

- **Description:**

Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.

- **Detection results:**

PASSED!

- **Security suggestion:**
no.

Permission restrictions

- **Description:**

Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.

• Detection results:

PASSED!

• Security suggestion:

no.



The background is a dark teal color with a complex, layered geometric pattern. In the center, there is a 3D cube with a blue base and a teal top. Above the cube is a transparent teal rectangular prism. The background is filled with floating binary code (0s and 1s) and two large, stylized teal shields on the left and right sides. The shields have a grid-like pattern and are slightly tilted.

armors.io

contact@armors.io

