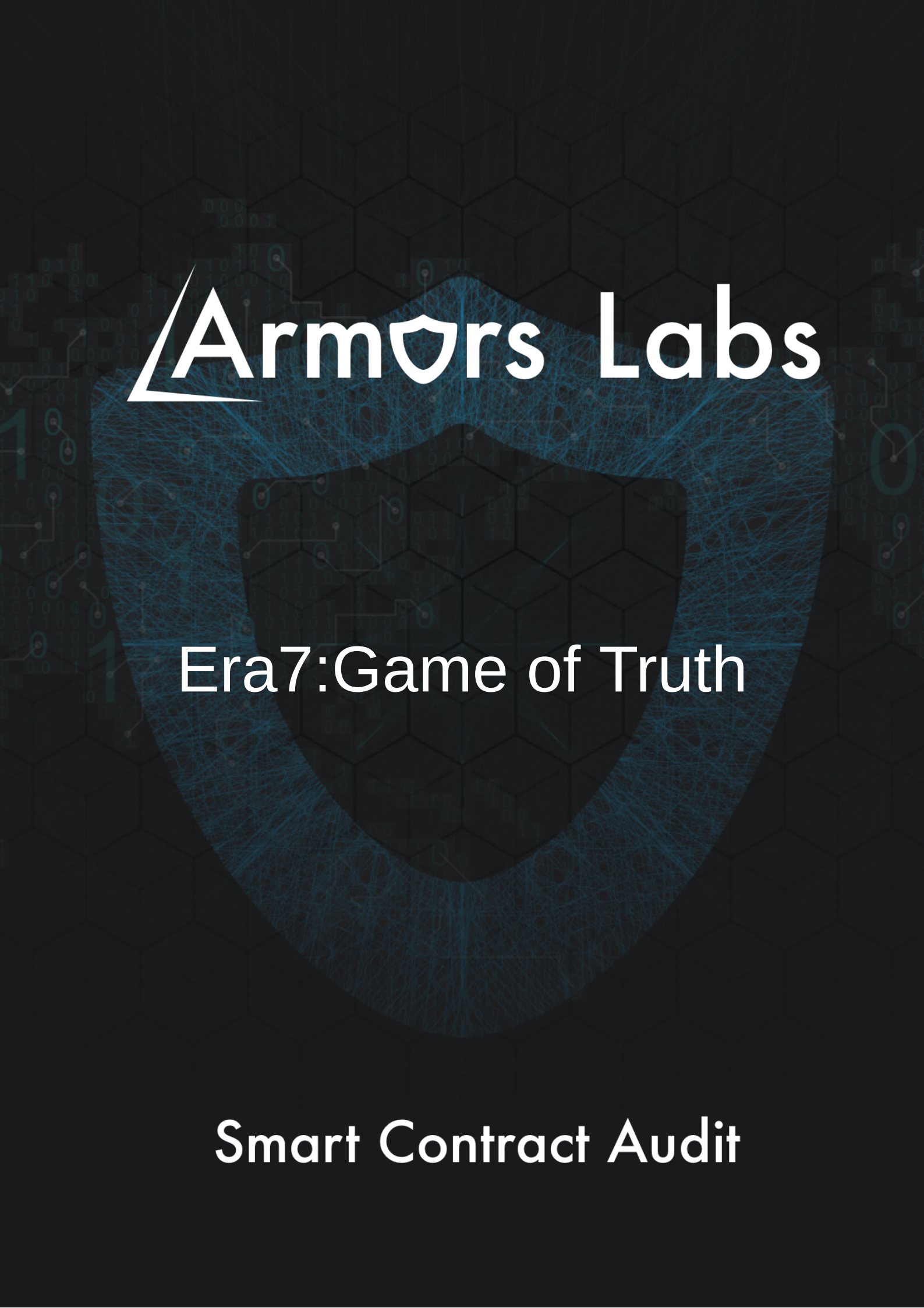

Armors Labs
Era7:Game of Truth
Smart Contract Audit

Era7:Game of Truth Audit Summary

Project name : Era7:Game of Truth Contract

Project address: None

Code URL : https://github.com/Era7Game/contracts

Commit : a3d6726a8e8ea57a476e606dde0e9efd8dd5af13

Project target : Era7:Game of Truth Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202111190006

Audit Team : Armors Labs

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

Era7:Game of Truth Audit

The Era7:Game of Truth team asked us to review and audit their Era7:Game of Truth 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
Era7:Game of Truth Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-11-19

Audit results

Note that:

1. FOTCard 721 The status of allCards was not updated during token destruction, bug the project side confirms that this is a record only.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Era7:Game of Truth 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.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

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. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
./IERA7Card.sol	a44cb4957b4a6ff874ea40e4a8018d8b
./comm/SafeMath.sol	3a3a90c14f63931c53884c2d080d90f8
./comm/Helper.sol	ce63bfc689299301327f75bbbb896ef4
./ERA7Token.sol	e6dfc6ec64710a3ab770a061cd03a4ee
./ERA7CardPreSale.sol	39abd762d7c8217bf6cb9cf58a6459f4
./ERA7CardMarketPlace.sol	0fc4427895618b355f42568ea5e1be4c
./ERA7Card.sol	7c30c920a3be100f6a94d6b3f96bf0e9

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

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

Contract file

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "./comm/Helper.sol";

contract ERA7Card is ERC721,Helper {

  struct ERA7CardEntity {
    uint256 tokenId;
    uint id;
    uint ct;
  }

  ERA7CardEntity[] public allCards;
  mapping(address => uint256[]) public playerCards;
  mapping(address => mapping(uint256 => uint)) public playerCardIndexs;

  using Counters for Counters.Counter;
  Counters.Counter private _tokenIds;

  constructor() ERC721("Era7 NFT", "ERANFT") {}

  function awardCard(address player,uint cardId) external onlyHelper returns (uint256){
    _tokenIds.increment();

    uint256 newItemId = _tokenIds.current();
```

```
    ERA7CardEntity memory card = ERA7CardEntity(newItemId,cardId,block.timestamp);
    allCards.push(card);

    playerCards[player].push(newItemId);
    playerCardIndexs[player][newItemId] = playerCards[player].length;

    _mint(player, newItemId);

    emit AwardCard(player,newItemId,cardId);

    return newItemId;
  }

  function approveList(address to, uint256[] memory tokenIds) external {
    uint len = tokenIds.length;
    for(uint i = 0; i < len ; i++){
      approve(to, tokenIds[i]);
    }
  }

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

  function burnList(uint256[] memory tokenIds) external {
    uint len = tokenIds.length;
    for(uint i = 0; i < len ; i++){
      _burn(tokenIds[i]);
    }
  }

  function burn(uint256 tokenId) external {
    _burn(tokenId);
  }

  function _burn(uint256 tokenId) internal virtual override {
    address owner = ERC721.ownerOf(tokenId);
    _swapTokenOwner(owner,address(0),tokenId);
    super._burn(tokenId);
  }

  function _swapTokenOwner(address from,address to,uint256 tokenId) private{
    if(from != to){
      uint index = playerCardIndexs[from][tokenId];
      if(playerCards[from].length != index){
        uint256 oldToken = playerCards[from][playerCards[from].length - 1];
        playerCards[from][index - 1] = oldToken;
        playerCardIndexs[from][oldToken] = index;
      }
      playerCards[from].pop();
      delete playerCardIndexs[from][tokenId];

      if(to != address(0)){
        playerCards[to].push(tokenId);
        playerCardIndexs[to][tokenId] = playerCards[to].length;
      }
    }
  }

  function totalCard() external view returns(uint256) {
    return allCards.length;
  }

  function getPlayerCards(address player) external view returns (ERA7CardEntity[] memory) {
```

```
    uint[] memory list = playerCards[player];
    uint length = list.length;
    ERA7CardEntity[] memory cardList = new ERA7CardEntity[](length);
    for(uint i = 0; i < length ; i++){
      cardList[i] = allCards[list[i] - 1];
    }
    return cardList;
  }

  event AwardCard(address indexed to, uint256 nftId,uint256 cardId);
}
// nft交易市场
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "./comm/SafeMath.sol";
import "./comm/Helper.sol";
import "./IERA7Card.sol";

contract ERA7CardMarketPlace is Helper {

  address public coin;
  address public nft;

  uint256[] public sellingNfts;
  mapping(uint256 => uint256) public sellingNftIndexs;

  struct MarketPlaceNftInfo{
    uint256 tokenId;
    uint price;
    address owner;
    uint upTime;
  }
  mapping(uint256 => MarketPlaceNftInfo) public nftMap;

  constructor() {}

  function initAddress(address coin_,address nft_) external onlyHelper {
    require((coin_ != address(0) && nft_ != address(0)), "ERA7CardMarketPlace initAddress: address er

    coin = coin_;
    nft = nft_;
  }

  function withdraw(address taxWallet) external onlyHelper returns(bool){
    require(taxWallet != address(0), "ERA7CardMarketPlace withdraw: taxWallet error");

    uint256 val = IERC20(coin).balanceOf(address(this));
    require(val > 0, "ERA7CardMarketPlace withdraw: val error");

    IERC20(coin).transfer(taxWallet,val);
    return true;
  }

  function uploadNft(uint256 nftId,uint256 price) external nonReentrant isPause{
      address ownerAddress = IERA7Card(nft).ownerOf(nftId);
      address uploadAddress = _msgSender();
      require(ownerAddress == uploadAddress, "ERA7CardMarketPlace uploadNft: not owner");
      require(price > 10000, "ERA7CardMarketPlace uploadNft: price error");

      uint index = sellingNftIndexs[nftId];
      if(index == 0){
        MarketPlaceNftInfo memory newInfo = MarketPlaceNftInfo(nftId,price,ownerAddress,block.timesta
        nftMap[nftId] = newInfo;
        sellingNfts.push(nftId);
```

```
        sellingNftIndexs[nftId] = sellingNfts.length;
      }else{
        MarketPlaceNftInfo storage oldInfo = nftMap[nftId];
        oldInfo.price = price;
        oldInfo.upTime = block.timestamp;
      }

      emit UploadNft(ownerAddress,nftId,price);
  }

  function stopSell(uint256 nftId) external nonReentrant isPause{
    uint index = sellingNftIndexs[nftId];
    require(index > 0, "ERA7CardMarketPlace stopSell: nftId error");

    address ownerAddress = IERA7Card(nft).ownerOf(nftId);
    require(ownerAddress == _msgSender(), "ERA7CardMarketPlace stopSell: stop error");

    _removeNftFromList(nftId);

    emit StopSell(nftId);
  }

  function _removeNftFromList(uint256 nftId) private{
    uint index = sellingNftIndexs[nftId];
      if(sellingNfts.length != index){
        uint oldNftId = sellingNfts[sellingNfts.length - 1];
        sellingNfts[index - 1] = oldNftId;
        sellingNftIndexs[oldNftId] = index;
      }
      sellingNfts.pop();
      delete sellingNftIndexs[nftId];
      delete nftMap[nftId];
  }

  function getTotalNft() external view returns(uint){
    return sellingNfts.length;
  }

  function getSellList(uint start,uint end) external view returns(MarketPlaceNftInfo[] memory){
      require(start >= 0 && end >= start,"ERA7CardMarketPlace getSellList:params error");

      uint total = sellingNfts.length;
      if(total == 0){
        return new MarketPlaceNftInfo[](0);
      }
      if(start >= total){
        start = total - 1;
      }
      if(end >= total){
        end = total - 1;
      }
      uint size = end - start;
      require(size <= 100,"ERA7CardMarketPlace getSellList:size error");

      MarketPlaceNftInfo[] memory list = new MarketPlaceNftInfo[](size + 1);
      for(uint i = start; i <= end ; i++){
        list[i - start] = nftMap[sellingNfts[i]];
      }
      return list;
  }

  function buy(uint256 nftId) external nonReentrant isPause {
    MarketPlaceNftInfo memory info = nftMap[nftId];
    require(info.tokenId > 0, "ERA7CardMarketPlace buy: nftId error");

    address ownerAddress = IERA7Card(nft).ownerOf(nftId);
```

```
    require(ownerAddress == info.owner, "ERA7CardMarketPlace buy: nftId owner change");

    SafeERC20.safeTransferFrom(IERC20(coin),_msgSender(),address(this),info.price);
    IERA7Card(nft).transferFrom(info.owner,address(this),nftId);

    uint256 get = SafeMath.mul(SafeMath.div(info.price,100),95);
    SafeERC20.safeTransfer(IERC20(coin),info.owner,get);
    IERA7Card(nft).transferFrom(address(this),_msgSender(),nftId);

    _removeNftFromList(nftId);

    emit Buy(info.owner,_msgSender(),info.tokenId,info.price);
  }

  event UploadNft(address indexed from, uint256 nftId,uint256 price);
  event StopSell(uint256 nftId);
  event Buy(address indexed from, address indexed to, uint256 nftId,uint256 price);
```



```
}
// Nft 预售合约
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "./comm/Helper.sol";
import "./IERA7Card.sol";

contract ERA7CardPreSale is Helper {

  address public coin;
  address public nft;

  struct PreSaleNftInfo{
    uint cardId;
    uint price;
    uint count;
  }

  uint[] public sellingNfts;
  mapping(uint => uint) public sellingNftIndexs;
  mapping(uint => PreSaleNftInfo) public sellingInfos;

  constructor() {}

  function initAddress(address coin_,address nft_) external onlyHelper {
    require((coin_ != address(0) && nft_ != address(0)), "ERA7CardPreSale initAddress: address error"

    coin = coin_;
    nft = nft_;
  }

  function withdraw(address taxWallet) external onlyHelper returns(bool){
```

```
    require(taxWallet != address(0), "ERA7CardPreSale withdraw: taxWallet error");

    uint256 val = IERC20(coin).balanceOf(address(this));
    require(val > 0, "ERA7CardPreSale withdraw: val error");

    IERC20(coin).transfer(taxWallet,val);
    return true;
  }

  function uploadNft(uint cardId,uint256 price,uint count) external onlyHelper{
      require(cardId > 0 && price > 0 && count > 0, "ERA7CardPreSale uploadNft: params error");

      uint index = sellingNftIndexs[cardId];
      if(index == 0){
        PreSaleNftInfo memory newInfo = PreSaleNftInfo(cardId,price,count);
        sellingInfos[cardId] = newInfo;
        sellingNfts.push(cardId);
        sellingNftIndexs[cardId] = sellingNfts.length;
      }else{
        PreSaleNftInfo storage oldInfo = sellingInfos[cardId];
        oldInfo.price = price;
        oldInfo.count = count;
      }
  }

  function stopSell(uint cardId) external onlyHelper{
      PreSaleNftInfo storage info = sellingInfos[cardId];
      require(info.cardId > 0, "ERA7CardPreSale stopSell: cardId error");

      uint index = sellingNftIndexs[cardId];
      if(sellingNfts.length != index){
        uint oldCardId = sellingNfts[sellingNfts.length - 1];
        sellingNfts[index - 1] = oldCardId;
        sellingNftIndexs[oldCardId] = index;
      }
      sellingNfts.pop();
      delete sellingNftIndexs[cardId];
      delete sellingInfos[cardId];
  }

  function getSellList() external view returns(PreSaleNftInfo[] memory){
      uint len = sellingNfts.length;
      PreSaleNftInfo[] memory list = new PreSaleNftInfo[](len);
      for(uint i = 0; i < len ;i++){
        list[i] = sellingInfos[sellingNfts[i]];
      }
      return list;
  }

  function buy(uint cardId) external nonReentrant returns(uint256){
      PreSaleNftInfo storage info = sellingInfos[cardId];
      require(info.cardId > 0, "ERA7CardPreSale buy: cardId error");

      uint count = info.count;
      require(count > 0, "ERA7CardPreSale buy: count error");
      info.count--;

      address buyAddress = _msgSender();
      SafeERC20.safeTransferFrom(IERC20(coin),buyAddress,address(this),info.price);

      uint256 tokenId = IERA7Card(nft).awardCard(buyAddress,cardId);

      emit Buy(_msgSender(),cardId,info.price);

      return tokenId;
  }
```

```
  event Buy(address indexed to, uint256 cardId,uint256 price);
```

```
}
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./comm/SafeMath.sol";
import "./comm/Helper.sol";

contract ERA7Token is ERC20,Helper {

  uint256 public maxMint;

  constructor() ERC20("Era Token", "ERA") {
    uint256 decimal = 10 ** uint256(decimals());
    maxMint = 1000000000 * decimal;
  }

  function mint(address to,uint256 amount) external onlyHelper {
    require(to != address(0), "ERA7Token:to address error");

    uint256 newVal = SafeMath.add(amount,totalSupply());
    require(newVal <= maxMint, "ERA7Token:mint value is max");
    _mint(to,amount);
  }


  function burn(uint256 amount) external {
      _burn(_msgSender(), amount);
  }

  function burnFrom(address account, uint256 amount) external {
      uint256 currentAllowance = allowance(account, _msgSender());
      require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
      unchecked {
          _approve(account, _msgSender(), currentAllowance - amount);
      }
      _burn(account, amount);
  }

}
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract Helper is Ownable,ReentrancyGuard {

  address[] public helpers;
  mapping(address => uint) helperIndexs;
```

```
  bool public pause;

  constructor(){
    pause = false;
  }

  function addHelper(address helper) external onlyOwner {
    require(helper != address(0),"Helper:set helper error");
    uint index = helperIndexs[helper];
    if(index == 0){
      helpers.push(helper);
      helperIndexs[helper] = helpers.length;
    }
  }

  function removeHelper(address helper) external onlyOwner{
    uint index = helperIndexs[helper];
    require(index > 0,"Helper:remove helper error");
    if(helpers.length != index){
        address old = helpers[helpers.length - 1];
        helpers[index - 1] = old;
        helperIndexs[old] = index;
      }
      helpers.pop();
      delete helperIndexs[helper];
  }

  function pauseContract() external onlyHelper{
    pause = true;
  }

  function resume() external onlyHelper{
    pause = false;
  }

  modifier onlyHelper() {
    require(helperIndexs[_msgSender()] > 0 || owner() == _msgSender(), "Helper: caller is not the hel
    _;
  }

  modifier isPause() {
    require(!pause, "Helper: contract is paused");
    _;
  }

}
// SPDX-License-Identifier: MIT
pragma solidity ^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 an
 * error, which is the standard behavior in high level programming languages.
 * SafeMath restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
```

```
     * _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 substraction of two unsigned integers, with an overflow flag.
     *
     * _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 overflow flag.
     *
     * _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 a division by zero flag.
     *
     * _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 a division by zero flag.
     *
     * _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 modulo),
     * 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 on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * 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 on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * 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 modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * 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;
    }
}// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface IERA7Card {
    struct ERA7CardEntity {
        uint256 tokenId;
```

```
        uint id;
        uint ct;
    }
    function balanceOf(address account) external view returns (uint256);
    function ownerOf(uint256 tokenId) external view returns (address owner);
    function transferFrom(address from,address to,uint256 tokenId) external;
    function awardCard(address player,uint cardId) external returns (uint256);
    function getAllCards() external view returns(ERA7CardEntity[] memory);
}
```

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 whBlockchain Currency 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 devestating vulernabilities 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 intialising 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!
```

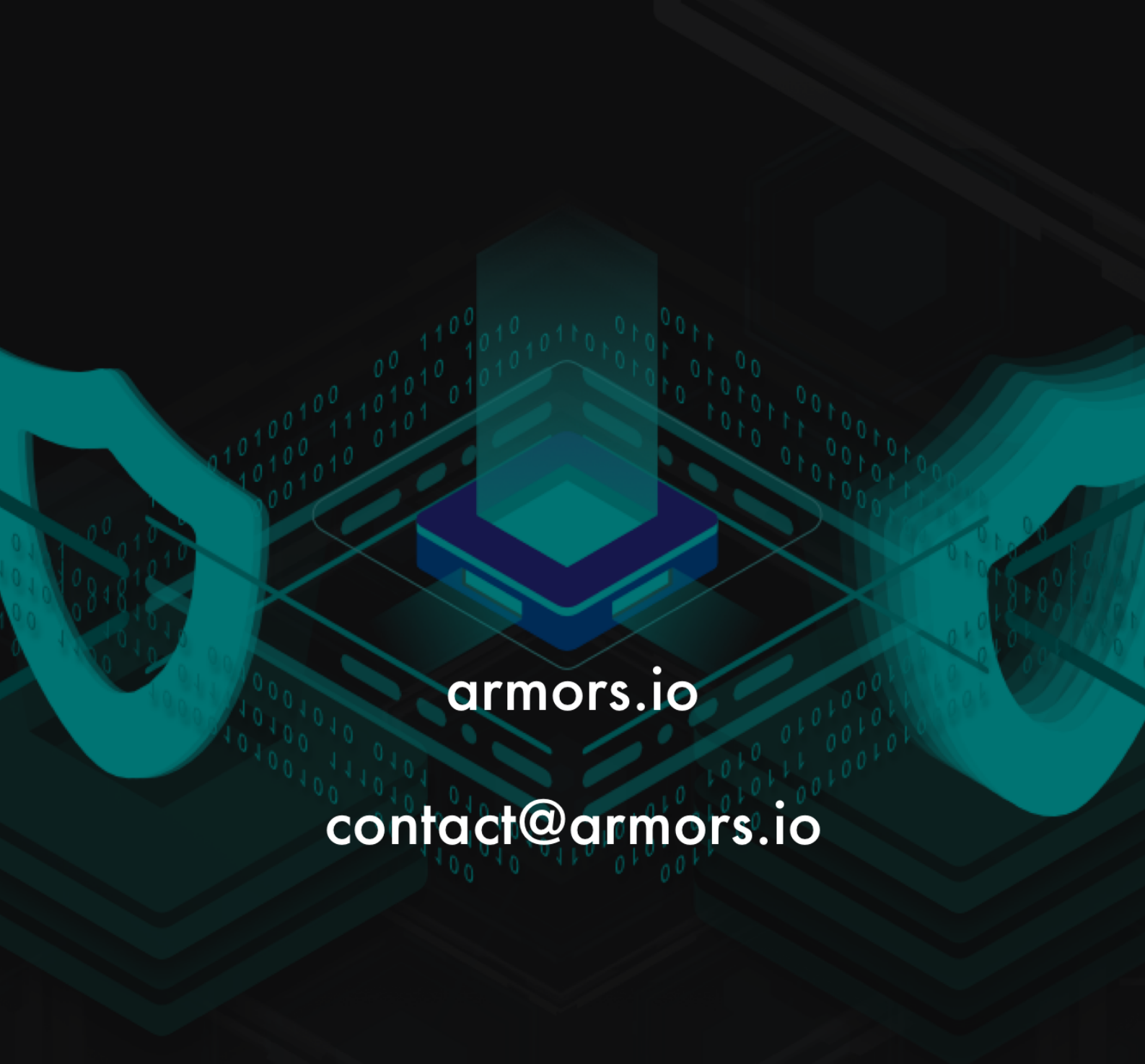
- **Security suggestion:**
  no.

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.



armors.io

contact@armors.io