Armors Labs

AiSwap

Smart Contract Audit

AiSwap Audit Summary

Project name : AiSwap Contract

Project address: None

Code URL : https://www.oklink.com/okexchain/address/0xc799b45d8b529bb7544de1fb24ba88401649dbce

Code URL : https://www.oklink.com/okexchain/address/0xd1303acaaed09911a36b41f57387d12159324466

Code URL : https://www.oklink.com/okexchain/address/0x7457197c455fcf4d454df2c06ecd05dbf25efb92

Commit : None

Project target : AiSwap Contract Audit

Blockchain : OKExChain

Test result : PASSED

Audit Info

Audit NO : 0X202105130006

Audit Team : Armors Labs

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

AiSwap Audit

The AiSwap team asked us to review and audit their AiSwap 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
AiSwap Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-05-13

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the AiSwap 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
AiSwapFactory	0xc799b45d8b529bb7544de1fb24ba88401649dbce
AiSwapPair	0xd1303acaaed09911a36b41f57387d12159324466
AiSwapRouter02	0x7457197c455fcf4d454df2c06ecd05dbf25efb92

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

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

Contract file

```
pragma solidity =0.5.16;

/**
 * @title Initializable
 *
 * @dev Helper contract to support initializer functions. To use it, replace
 * the constructor with a function that has the `initializer` modifier.
 * WARNING: Unlike constructors, initializer functions must be manually
 * invoked. This applies both to deploying an Initializable contract, as well
 * as extending an Initializable contract via inheritance.
 * WARNING: When used with inheritance, manual care must be taken to not invoke
 * a parent initializer twice, or ensure that all initializers are idempotent,
 * because this is not dealt with automatically as with constructors.
 */
contract Initializable {

  /**
   * @dev Indicates that the contract has been initialized.
   */
  bool private initialized;

  /**
   * @dev Indicates that the contract is in the process of being initialized.
   */
  bool private initializing;

  /**
   * @dev Modifier to use in the initializer function of a contract.
   */
  modifier initializer() {
    require(initializing || isConstructor() || !initialized, "Contract instance has already been init

    bool isTopLevelCall = !initializing;
    if (isTopLevelCall) {
      initializing = true;
      initialized = true;
    }

    _;

    if (isTopLevelCall) {
      initializing = false;
    }
  }

  /// @dev Returns true if and only if the function is running in the constructor
  function isConstructor() private view returns (bool) {
    // extcodesize checks the size of the code stored in an address, and
    // address returns the current address. Since the code is still not
    // deployed when running a constructor, any checks on its code size will
    // yield zero, making it an effective way to detect if a contract is
    // under construction or not.
    address self = address(this);
    uint256 cs;
    assembly { cs := extcodesize(self) }
    return cs == 0;
  }

  // Reserved storage space to allow for layout changes in the future.
  uint256[50] private ______gap;
}


contract Governable is Initializable {
    address public governor;

    event GovernorshipTransferred(address indexed previousGovernor, address indexed newGovernor);

    /**
     * @dev Contract initializer.
     * called once by the factory at time of deployment
     */
    function initialize(address governor_) public initializer {     // virtual
        governor = governor_;
        emit GovernorshipTransferred(address(0), governor);
    }

    modifier governance() {
        require(msg.sender == governor);
        _;
    }

    /**
     * @dev Allows the current governor to relinquish control of the contract.
     * @notice Renouncing to governorship will leave the contract without an governor.
     * It will not be possible to call the functions with the `governance`
     * modifier anymore.
     */
    function renounceGovernorship() public governance {
        emit GovernorshipTransferred(governor, address(0));
        governor = address(0);
    }

    /**
     * @dev Allows the current governor to transfer control of the contract to a newGovernor.
     * @param newGovernor The address to transfer governorship to.
     */
    function transferGovernorship(address newGovernor) public governance {
        _transferGovernorship(newGovernor);
    }

    /**
     * @dev Transfers control of the contract to a newGovernor.
     * @param newGovernor The address to transfer governorship to.
     */
    function _transferGovernorship(address newGovernor) internal {
        require(newGovernor != address(0));
        emit GovernorshipTransferred(governor, newGovernor);
        governor = newGovernor;
    }
}


contract Configurable is Governable {
    mapping (bytes32 => uint) internal config;

    function getConfig(bytes32 key) public view returns (uint) {
        return config[key];
    }
    function getConfig(bytes32 key, uint index) public view returns (uint) {
        return config[bytes32(uint(key) ^ index)];
    }
    function getConfig(bytes32 key, address addr) public view returns (uint) {
        return config[bytes32(uint(key) ^ uint(addr))];
    }

    function _setConfig(bytes32 key, uint value) internal {
        if(config[key] != value)
            config[key] = value;
    }
    function _setConfig(bytes32 key, uint index, uint value) internal {
        _setConfig(bytes32(uint(key) ^ index), value);
    }
    function _setConfig(bytes32 key, address addr, uint value) internal {
        _setConfig(bytes32(uint(key) ^ uint(addr)), value);
    }

    function setConfig(bytes32 key, uint value) external governance {
        _setConfig(key, value);
    }
    function setConfig(bytes32 key, uint index, uint value) external governance {
        _setConfig(bytes32(uint(key) ^ index), value);
    }
    function setConfig(bytes32 key, address addr, uint value) public governance {
        _setConfig(bytes32(uint(key) ^ uint(addr)), value);
    }
}


interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
}

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);
    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

interface IUniswapV2ERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by
}

interface IERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function decimals() external view returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);
}

interface IUniswapV2Callee {
    function uniswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) external;
}

contract UniswapV2ERC20 is IUniswapV2ERC20, Initializable {
    using SafeMath for uint;

    string public constant name = 'AiSwap Liquidity Provider Token';
    string public constant symbol = 'ALPT';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)"
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c6
    mapping(address => uint) public nonces;

    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    constructor() public {
        initialize();
    }

    function initialize() internal initializer {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingC
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }

    function _burn(address from, uint value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address from, address to, uint value) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint value) external returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(address from, address to, uint value) external returns (bool) {
        if (allowance[from][msg.sender] != uint(-1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        }
        _transfer(from, to, value);
        return true;
    }

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by
        require(deadline >= block.timestamp, 'UniswapV2: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadlin
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'UniswapV2: INVALID_SIGN
        _approve(owner, spender, value);
    }
}

contract AiSwapPair is IUniswapV2Pair, UniswapV2ERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves

    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity even

    //uint private unlocked = 1;
    //modifier lock() {
    //    require(unlocked == 1, 'UniswapV2: LOCKED');
    //    unlocked = 0;
    //    _;
    //    unlocked = 1;
    //}
    // compatibility for Upgradeable, as described in
    // https://docs.openzeppelin.com/upgrades/2.8/writing-upgradeable#avoid-initial-values-in-field-d
    uint private locked;
    modifier lock() {
        require(locked == 0, 'AiSwap: LOCKED');
        locked = 1;
        _;
        locked = 0;
    }

    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTi
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'UniswapV2: TRANSFER_FAILE
    }

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    constructor() public {
        factory = msg.sender;
    }

    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external initializer {
        //require(msg.sender == factory, 'UniswapV2: FORBIDDEN'); // sufficient check
        UniswapV2ERC20.initialize();
        factory = msg.sender;
        token0 = _token0;
        token1 = _token1;
    }

    // update reserves and, on the first call per block, price accumulators
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
        require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'UniswapV2: OVERFLOW');
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
            price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
            price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
        }
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }

    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
        address feeTo = IUniswapV2Factory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint _kLast = kLast; // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = Math.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator = rootK.mul(5).add(rootKLast);
                    uint liquidity = numerator / denominator;
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }

    // this low-level function should be called from a contract which performs important safety check
    function mint(address to) external lock returns (uint liquidity) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can u
        if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
           _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY toke
        } else {
            liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _
        }
        require(liquidity > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Mint(msg.sender, amount0, amount1);
    }

    // this low-level function should be called from a contract which performs important safety check
    function burn(address to) external lock returns (uint amount0, uint amount1) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        address _token0 = token0;                                // gas savings
        address _token1 = token1;                                // gas savings
        uint balance0 = IERC20(_token0).balanceOf(address(this));
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can u
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribu
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribu
        require(amount0 > 0 && amount1 > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Burn(msg.sender, amount0, amount1, to);
    }
    // this low-level function should be called from a contract which performs important safety check
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY'

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
        address _token0 = token0;
        address _token1 = token1;
        require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
        if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
        if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
        if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, d
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2)
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)), res
    }
}


/**
 * @title Proxy
 * @dev Implements delegation of calls to other contracts, with proper
 * forwarding of return values and bubbling of failures.
 * It defines a fallback function that delegates all calls to the address
 * returned by the abstract _implementation() internal function.
 */
contract Proxy {
  /**
   * @dev Fallback function.
   * Implemented entirely in `_fallback`.
   */
  function () payable external {
    _fallback();
  }

  /**
   * @return The Address of the implementation.
   */
  function _implementation() internal view returns (address);

  /**
   * @dev Delegates execution to an implementation contract.
   * This is a low level function that doesn't return to its internal call site.
   * It will return to the external caller whatever the implementation returns.
   * @param implementation Address to delegate.
   */
  function _delegate(address implementation) internal {
    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 Function that is run as the first thing in the fallback function.
   * Can be redefined in derived contracts to add functionality.
   * Redefinitions must call super._willFallback().
   */
  function _willFallback() internal {
  }

  /**
   * @dev fallback implementation.
   * Extracted to enable manual triggering.
   */
  function _fallback() internal {
    if(OpenZeppelinUpgradesAddress.isContract(msg.sender) && msg.data.length == 0 && gasleft() <= 230
        return;
    _willFallback();
    _delegate(_implementation());
  }
}


/**
 * @title BaseUpgradeabilityProxy
 * @dev This contract implements a proxy that allows to change the
 * implementation address to which it will delegate.
 * Such a change is called an implementation upgrade.
 */
contract BaseUpgradeabilityProxy is Proxy {
  /**
   * @dev Emitted when the implementation is upgraded.
   * @param implementation Address of the new implementation.
   */
  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 is
```

```
     * validated in the constructor.
     */
    bytes32 internal constant IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a92

    /**
     * @dev Returns the current implementation.
     * @return Address of the current implementation
     */
    function _implementation() internal view returns (address impl) {
      bytes32 slot = IMPLEMENTATION_SLOT;
      assembly {
        impl := sload(slot)
      }
    }

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

    /**
     * @dev Sets the implementation address of the proxy.
     * @param newImplementation Address of the new implementation.
     */
    function _setImplementation(address newImplementation) internal {
      require(OpenZeppelinUpgradesAddress.isContract(newImplementation), "Cannot set a proxy implementa

      bytes32 slot = IMPLEMENTATION_SLOT;

      assembly {
        sstore(slot, newImplementation)
      }
    }
  }


  /**
   * @title BaseAdminUpgradeabilityProxy
   * @dev This contract combines an upgradeability proxy with an authorization
   * mechanism for administrative tasks.
   * All external functions in this contract must be guarded by the
   * `ifAdmin` modifier. See ethereum/solidity#3864 for a Solidity
   * feature proposal that would enable this to be done automatically.
   */
  contract BaseAdminUpgradeabilityProxy is BaseUpgradeabilityProxy {
    /**
     * @dev Emitted when the administration has been transferred.
     * @param previousAdmin Address of the previous admin.
     * @param newAdmin Address of the new admin.
     */
    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 internal constant ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b

    /**
     * @dev Modifier to check whether the `msg.sender` is the admin.
   * If it is, it will run the function. Otherwise, it will delegate the call
   * to the implementation.
   */
  modifier ifAdmin() {
    if (msg.sender == _admin()) {
      _;
    } else {
      _fallback();
    }
  }

  /**
   * @return The address of the proxy admin.
   */
  function admin() external ifAdmin returns (address) {
    return _admin();
  }

  /**
   * @return The address of the implementation.
   */
  function implementation() external ifAdmin returns (address) {
    return _implementation();
  }

  /**
   * @dev Changes the admin of the proxy.
   * Only the current admin can call this function.
   * @param newAdmin Address to transfer proxy administration to.
   */
  function changeAdmin(address newAdmin) external ifAdmin {
    require(newAdmin != address(0), "Cannot change the admin of a proxy to the zero address");
    emit AdminChanged(_admin(), newAdmin);
    _setAdmin(newAdmin);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy.
   * Only the admin can call this function.
   * @param newImplementation Address of the new implementation.
   */
  function upgradeTo(address newImplementation) external ifAdmin {
    _upgradeTo(newImplementation);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy and call a function
   * on the new implementation.
   * This is useful to initialize the proxied contract.
   * @param newImplementation Address of the new implementation.
   * @param data Data to send as msg.data in the low level call.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   */
  function upgradeToAndCall(address newImplementation, bytes calldata data) payable external ifAdmin
    _upgradeTo(newImplementation);
    (bool success,) = newImplementation.delegatecall(data);
    require(success);
  }

  /**
   * @return The admin slot.
   */
  function _admin() internal view returns (address adm) {
    bytes32 slot = ADMIN_SLOT;
    assembly {
      adm := sload(slot)
    }
  }

  /**
   * @dev Sets the address of the proxy admin.
   * @param newAdmin Address of the new proxy admin.
   */
  function _setAdmin(address newAdmin) internal {
    bytes32 slot = ADMIN_SLOT;

    assembly {
      sstore(slot, newAdmin)
    }
  }

  /**
   * @dev Only fall back when the sender is not the admin.
   */
  function _willFallback() internal {
    require(msg.sender != _admin(), "Cannot call fallback function from the proxy admin");
    super._willFallback();
  }
}


/**
 * @title UpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with a constructor for initializing
 * implementation and init data.
 */
contract UpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Contract constructor.
   * @param _logic Address of the initial implementation.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  constructor(address _logic, bytes memory _data) public payable {
    assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
    _setImplementation(_logic);
    if(_data.length > 0) {
      (bool success,) = _logic.delegatecall(_data);
      require(success);
    }
  }
}


/**
 * @title AdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with a constructor for
 * initializing the implementation, admin, and init data.
 */
contract AdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy, UpgradeabilityProxy {
  /**
   * Contract constructor.
   * @param _logic address of the initial implementation.
   * @param _admin Address of the proxy administrator.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  constructor(address _logic, address _admin, bytes memory _data) UpgradeabilityProxy(_logic, _data)
    assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
    _setAdmin(_admin);
  }
}


/**
 * @title InitializableUpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with an initializer for initializing
 * implementation and init data.
 */
contract InitializableUpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Contract initializer.
   * @param _logic Address of the initial implementation.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  function initialize(address _logic, bytes memory _data) public payable {
    require(_implementation() == address(0));
    assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
    _setImplementation(_logic);
    if(_data.length > 0) {
      (bool success,) = _logic.delegatecall(_data);
      require(success);
    }
  }
}


/**
 * @title InitializableAdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with an initializer for
 * initializing the implementation, admin, and init data.
 */
contract InitializableAdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy, InitializableUpgradea
  /**
   * Contract initializer.
   * @param _logic address of the initial implementation.
   * @param _admin Address of the proxy administrator.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  function initialize(address _logic, address _admin, bytes memory _data) public payable {
    require(_implementation() == address(0));
    InitializableUpgradeabilityProxy.initialize(_logic, _data);
    assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
    _setAdmin(_admin);
  }
}


interface IProxyFactory {
    function productImplementation() external view returns (address);
}


/**
 * @title ProductProxy
 * @dev This contract implements a proxy that
 * it is deploied by ProxyFactory,
 * and it's implementation is stored in factory.
 */
contract ProductProxy is Proxy {

  /**
   * @dev Storage slot with the address of the ProxyFactory.
   * This is the keccak-256 hash of "eip1967.proxy.factory" subtracted by 1, and is
   * validated in the constructor.
   */
  bytes32 internal constant FACTORY_SLOT = 0x7a45a402e4cb6e08ebc196f20f66d5d30e67285a2a8aa80503fa409e

  /**
   * @dev Sets the factory address of the ProductProxy.
   * @param newFactory Address of the new factory.
   */
  function _setFactory(address newFactory) internal {
    require(OpenZeppelinUpgradesAddress.isContract(newFactory), "Cannot set a factory to a non-contra

    bytes32 slot = FACTORY_SLOT;

    assembly {
      sstore(slot, newFactory)
    }
  }

  /**
   * @dev Returns the factory.
   * @return Address of the factory.
   */
  function _factory() internal view returns (address factory) {
    bytes32 slot = FACTORY_SLOT;
    assembly {
      factory := sload(slot)
    }
  }

  /**
   * @dev Returns the current implementation.
   * @return Address of the current implementation
   */
  function _implementation() internal view returns (address) {
    address factory = _factory();
    if(OpenZeppelinUpgradesAddress.isContract(factory))
        return IProxyFactory(factory).productImplementation();
    else
        return address(0);
  }

}


/**
 * @title InitializableProductProxy
 * @dev Extends ProductProxy with an initializer for initializing
 * factory and init data.
 */
contract InitializableProductProxy is ProductProxy {
  /**
   * @dev Contract initializer.
   * @param factory Address of the initial factory.
   * @param data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  function initialize(address factory, bytes memory data) public payable {
    require(_factory() == address(0));
    assert(FACTORY_SLOT == bytes32(uint256(keccak256('eip1967.proxy.factory')) - 1));
    _setFactory(factory);
    if(data.length > 0) {
      (bool success,) = _implementation().delegatecall(data);
      require(success);
    }
  }
}

/*
contract ProxyFactory {

  event ProxyCreated(address proxy);

  bytes32 private contractCodeHash;

  constructor() public {
    contractCodeHash = keccak256(
      type(InitializableAdminUpgradeabilityProxy).creationCode
    );
  }

  function deployMinimal(address _logic, bytes memory _data) public returns (address proxy) {
    // Adapted from https://github.com/optionality/clone-factory/blob/32782f82dfc5a00d103a7e61a17a5de
    bytes20 targetBytes = bytes20(_logic);
    assembly {
      let clone := mload(0x40)
      mstore(clone, 0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000000000000000000000)
      mstore(add(clone, 0x14), targetBytes)
      mstore(add(clone, 0x28), 0x5af43d82803e903d91602b57fd5bf30000000000000000000000000000000000)
      proxy := create(0, clone, 0x37)
    }

    emit ProxyCreated(address(proxy));

    if(_data.length > 0) {
      (bool success,) = proxy.call(_data);
      require(success);
    }
  }

  function deploy(uint256 _salt, address _logic, address _admin, bytes memory _data) public returns (
    return _deployProxy(_salt, _logic, _admin, _data, msg.sender);
  }

  function deploySigned(uint256 _salt, address _logic, address _admin, bytes memory _data, bytes memo
    address signer = getSigner(_salt, _logic, _admin, _data, _signature);
    require(signer != address(0), "Invalid signature");
    return _deployProxy(_salt, _logic, _admin, _data, signer);
  }

  function getDeploymentAddress(uint256 _salt, address _sender) public view returns (address) {
    // Adapted from https://github.com/archanova/solidity/blob/08f8f6bedc6e71c24758d20219b7d0749d7591
    bytes32 salt = _getSalt(_salt, _sender);
    bytes32 rawAddress = keccak256(
      abi.encodePacked(
        bytes1(0xff),
        address(this),
        salt,
        contractCodeHash
      )
    );

    return address(bytes20(rawAddress << 96));
  }
  function getSigner(uint256 _salt, address _logic, address _admin, bytes memory _data, bytes memory 
    bytes32 msgHash = OpenZeppelinUpgradesECDSA.toEthSignedMessageHash(
      keccak256(
        abi.encodePacked(
          _salt, _logic, _admin, _data, address(this)
        )
      )
    );

    return OpenZeppelinUpgradesECDSA.recover(msgHash, _signature);
  }

  function _deployProxy(uint256 _salt, address _logic, address _admin, bytes memory _data, address _s
    InitializableAdminUpgradeabilityProxy proxy = _createProxy(_salt, _sender);
    emit ProxyCreated(address(proxy));
    proxy.initialize(_logic, _admin, _data);
    return address(proxy);
  }

  function _createProxy(uint256 _salt, address _sender) internal returns (InitializableAdminUpgradeab
    address payable addr;
    bytes memory code = type(InitializableAdminUpgradeabilityProxy).creationCode;
    bytes32 salt = _getSalt(_salt, _sender);

    assembly {
      addr := create2(0, add(code, 0x20), mload(code), salt)
      if iszero(extcodesize(addr)) {
        revert(0, 0)
      }
    }

    return InitializableAdminUpgradeabilityProxy(addr);
  }

  function _getSalt(uint256 _salt, address _sender) internal pure returns (bytes32) {
    return keccak256(abi.encodePacked(_salt, _sender));
  }
}
*/

contract AiSwapFactory is IProxyFactory, IUniswapV2Factory {
    using SafeMath for uint;

    bytes32 public constant pairCodeHash = keccak256(abi.encodePacked(type(InitializableProductProxy)

    address public productImplementation;

    address public feeTo;
    address public feeToSetter;

    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function initialize(address _feeToSetter, address _productImplementation) public {
        require(feeToSetter== address(0) && _feeToSetter != address(0), 'AiSwapFactory.initialize can
        feeToSetter = _feeToSetter;
        productImplementation = _productImplementation;
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }
    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2: PAIR_EXISTS'); // single check is
        bytes memory bytecode = type(InitializableProductProxy).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        InitializableProductProxy(uint160(pair)).initialize(address(this), abi.encodeWithSignature('i
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }

    function setProductImplementation(address _productImplementation) public {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        productImplementation = _productImplementation;
    }

    // returns sorted token addresses, used to handle return values from pairs sorted in this order
    function sortTokens(address tokenA, address tokenB) public pure returns (address token0, address
        require(tokenA != tokenB, 'AiSwapFactory: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'AiSwapFactory: ZERO_ADDRESS');
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address tokenA, address tokenB) public view returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(uint(keccak256(abi.encodePacked(
                hex'ff',
                address(this),
                keccak256(abi.encodePacked(token0, token1)),
                pairCodeHash
            ))));
    }
    // fetches and sorts the reserves for a pair
    function getReserves(address tokenA, address tokenB) public view returns (uint reserveA, uint res
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IUniswapV2Pair(pairFor(tokenA, tokenB)).getReserves();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other ass
    function quote(uint amountA, uint reserveA, uint reserveB) public pure returns (uint amountB) {
        require(amountA > 0, 'AiSwapFactory: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'AiSwapFactory: INSUFFICIENT_LIQUIDITY');
        amountB = amountA.mul(reserveB) / reserveA;
    }

    // given an input amount of an asset and pair reserves, returns the maximum output amount of the
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) public pure returns (uint a
        require(amountIn > 0, 'AiSwapFactory: INSUFFICIENT_INPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'AiSwapFactory: INSUFFICIENT_LIQUIDITY');
        uint amountInWithFee = amountIn.mul(997);
        uint numerator = amountInWithFee.mul(reserveOut);
        uint denominator = reserveIn.mul(1000).add(amountInWithFee);
        amountOut = numerator / denominator;
    }

    // given an output amount of an asset and pair reserves, returns a required input amount of the o
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) public pure returns (uint a
        require(amountOut > 0, 'AiSwapFactory: INSUFFICIENT_OUTPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'AiSwapFactory: INSUFFICIENT_LIQUIDITY');
        uint numerator = reserveIn.mul(amountOut).mul(1000);
        uint denominator = reserveOut.sub(amountOut).mul(997);
        amountIn = (numerator / denominator).add(1);
    }

    // performs chained getAmountOut calculations on any number of pairs
    function getAmountsOut(uint amountIn, address[] memory path) public view returns (uint[] memory a
        require(path.length >= 2, 'AiSwapFactory: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[0] = amountIn;
        for (uint i; i < path.length - 1; i++) {
            (uint reserveIn, uint reserveOut) = getReserves(path[i], path[i + 1]);
            amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
        }
    }

    // performs chained getAmountIn calculations on any number of pairs
    function getAmountsIn(uint amountOut, address[] memory path) public view returns (uint[] memory a
        require(path.length >= 2, 'AiSwapFactory: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[amounts.length - 1] = amountOut;
        for (uint i = path.length - 1; i > 0; i--) {
            (uint reserveIn, uint reserveOut) = getReserves(path[i - 1], path[i]);
            amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
        }
    }
}


/**
 * @title Elliptic curve signature operations
 * @dev Based on https://gist.github.com/axic/5b33912c6f61ae6fd96d6c4a47afde6d
 * TODO Remove this library once solidity supports passing a signature to ecrecover.
 * See https://github.com/ethereum/solidity/issues/864
 *
 * Source https://raw.githubusercontent.com/OpenZeppelin/openzeppelin-solidity/79dd498b16b957399f84b9
 * This contract is copied here and renamed from the original to avoid clashes in the compiled artifa
 * when the user imports a zos-lib contract (that transitively causes this contract to be compiled an
 * build/artifacts folder) as well as the vanilla implementation from an openzeppelin version.
 */

library OpenZeppelinUpgradesECDSA {
    /**
     * @dev Recover signer address from a message by using their signature
     * @param hash bytes32 message, the hash is the signed message. What is recovered is the signer a
     * @param signature bytes signature, the signature is generated using web3.eth.sign()
     */
    function recover(bytes32 hash, bytes memory signature) internal pure returns (address) {
        // Check the signature length
        if (signature.length != 65) {
            return (address(0));
        }

        // Divide the signature in r, s and v variables
        bytes32 r;
        bytes32 s;
        uint8 v;

        // ecrecover takes the signature parameters, and the only way to get them
        // currently is to use assembly.
        // solhint-disable-next-line no-inline-assembly
        assembly {
            r := mload(add(signature, 0x20))
            s := mload(add(signature, 0x40))
            v := byte(0, mload(add(signature, 0x60)))
        }

        // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and mak
        // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/pa
        // the valid range for s in (281): 0 < s < secp256k1n ÷ 2 + 1, and for v in (282): v ∈ {27, 2
        // signatures from current libraries generate a unique signature with an s-value in the lower
        //
        // If your library generates malleable signatures, such as s-values in the upper range, calcu
        // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v fr
        // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27
        // these malleable signatures as well.
        if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
            return address(0);
        }

        if (v != 27 && v != 28) {
            return address(0);
        }

        // If the signature is valid (and not malleable), return the signer address
        return ecrecover(hash, v, r, s);
    }

    /**
     * toEthSignedMessageHash
     * @dev prefix a bytes32 value with "\x19Ethereum Signed Message:"
     * and hash the result
     */
    function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32) {
        // 32 is the length in bytes of hash,
        // enforced by the type signature above
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));
    }
}


/**
 * Utility library of inline functions on addresses
 *
 * Source https://raw.githubusercontent.com/OpenZeppelin/openzeppelin-solidity/v2.1.3/contracts/utils
 * This contract is copied here and renamed from the original to avoid clashes in the compiled artifa
 * when the user imports a zos-lib contract (that transitively causes this contract to be compiled an
 * build/artifacts folder) as well as the vanilla Address implementation from an openzeppelin version
 */
library OpenZeppelinUpgradesAddress {
    /**
     * Returns whether the target address is a contract
     * @dev This function will return false if invoked during the constructor of a contract,
     * as the code is not actually created until after the constructor finishes.
     * @param account address of the account to check
     * @return whether the target address is a contract
     */
    function isContract(address account) internal view returns (bool) {
        uint256 size;
        // XXX Currently there is no better way to check if there is a contract in an address
        // than to check the size of the code at that address.
        // See https://ethereum.stackexchange.com/a/14016/36603
        // for more details about how this works.
        // TODO Check this again before the Serenity release, because all addresses will be
        // contracts then.
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
}


// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-ma

library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }

    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format)

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
    uint224 constant Q112 = 2**112;

    // encode a uint112 as a UQ112x112
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112; // never overflows
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
```

```
}



interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadl
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address
        external
        returns (uint[] memory amounts);
    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address
        external
        returns (uint[] memory amounts);
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline
        external
        payable
        returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint
    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memo
    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memo
}

interface IUniswapV2Router02 {      // is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}

interface IWETH {
    function deposit() external payable;
    function transfer(address to, uint value) external returns (bool);
    function withdraw(uint) external;
}

contract AiSwapRouter02 is IUniswapV2Router01, IUniswapV2Router02, Initializable {
    using SafeMath for uint;

    address public factory;   // immutable override
    address public WETH;      // immutable override

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'UniswapV2Router: EXPIRED');
        _;
    }

    //constructor(address _factory, address _WETH) public {
    //    initialize(_factory, _WETH);
    //}

    function initialize(address _factory, address _WETH) public initializer {
        factory = _factory;
        WETH = _WETH;
    }

    function () external payable {  // receive
        assert(msg.sender == WETH); // only accept ETH via fallback from the WETH contract
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
            IUniswapV2Factory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = UniswapV2Library.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IUniswapV2Pair(pair).mint(to);
    }
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable ensure(deadline) returns (uint amountToken, uint amountETH, uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit.value(amountETH)();                                    // IWETH(WETH).de
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = IUniswapV2Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountA, uint amountB) {        // virtual override
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
        IUniswapV2Pair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint amount0, uint amount1) = IUniswapV2Pair(pair).burn(to);
        (address token0,) = UniswapV2Library.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
    }
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountToken, uint amountETH) {  // virtual override
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB) {                    // virtual override
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin, to, d
    }
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH) {              // virtual override
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin, amountETHMin,
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountETH) {                  // virtual override
        (, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountETH) {                                 // virtual override
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal {    // virtua
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = UniswapV2Library.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut,
            address to = i < path.length - 2 ? UniswapV2Library.pairFor(factory, output, path[i + 2])
            IUniswapV2Pair(UniswapV2Library.pairFor(factory, input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory amounts) {           // virtual override
        amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router: INSUFFICIENT_OUTPUT_AM
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory amounts) {           // virtual override
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'UniswapV2Router: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadl
        external
        //virtual
        //override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router: INSUFFICIENT_OUTPUT_AM
        IWETH(WETH).deposit.value(amounts[0])();                                    // IWETH(WETH).dep
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]))
        _swap(amounts, path, to);
    }
    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address
        external
        //virtual
        //override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'UniswapV2Router: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address
        external
        //virtual
        //override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router: INSUFFICIENT_OUTPUT_AM
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline
        external
        //virtual
        //override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'UniswapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'UniswapV2Router: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit.value(amounts[0])();                                        // IWETH(WETH)
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]))
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal {      /
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = UniswapV2Library.sortTokens(input, output);
            IUniswapV2Pair pair = IUniswapV2Pair(UniswapV2Library.pairFor(factory, input, output));
            uint amountInput;
            uint amountOutput;
            { // scope to avoid stack too deep errors
            (uint reserve0, uint reserve1,) = pair.getReserves();
            (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reser
            amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
            amountOutput = UniswapV2Library.getAmountOut(amountInput, reserveInput, reserveOutput);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amount0
            address to = i < path.length - 2 ? UniswapV2Library.pairFor(factory, output, path[i + 2])
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }
    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external ensure(deadline) {                                           // virtual override
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amountIn
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        //virtual
        //override
        payable
        ensure(deadline)
    {
        require(path[0] == WETH, 'UniswapV2Router: INVALID_PATH');
        uint amountIn = msg.value;
        IWETH(WETH).deposit.value(amountIn)();                                     // IWETH(WETH)
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0], path[1]), amountIn));
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        //virtual
        //override
        ensure(deadline)
    {
        require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WETH).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).withdraw(amountOut);
        TransferHelper.safeTransferETH(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(uint amountA, uint reserveA, uint reserveB) public pure returns (uint amountB) {
        return UniswapV2Library.quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
        public
        pure
        //virtual
        //override
        returns (uint amountOut)
    {
        return UniswapV2Library.getAmountOut(amountIn, reserveIn, reserveOut);
    }

    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
        public
        pure
        //virtual
        //override
        returns (uint amountIn)
    {
        return UniswapV2Library.getAmountIn(amountOut, reserveIn, reserveOut);
    }

    function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        //virtual
        //override
        returns (uint[] memory amounts)
    {
        return UniswapV2Library.getAmountsOut(factory, amountIn, path);
    }

    function getAmountsIn(uint amountOut, address[] memory path)
        public
        view
        //virtual
        //override
        returns (uint[] memory amounts)
    {
        return UniswapV2Library.getAmountsIn(factory, amountOut, path);
    }
}

library UniswapV2Library {
    using SafeMath for uint;
    bytes32 private constant PairCodeHash = keccak256(type(InitializableProductProxy).creationCode);
    //bytes32 private constant PairCodeHash = hex'9e3d176cd7b9504eb5f6b77283eeba7ad886f58601c2a02d5ad
    //bytes32 private constant PairCodeHash = hex'71d15772a2b431bfcb85fd38973fe760b5765f961d5586544c6

    function pairCodeHash() internal pure returns (bytes32) {
        return PairCodeHash;
    }
```

```
// returns sorted token addresses, used to handle return values from pairs sorted in this order
function sortTokens(address tokenA, address tokenB) internal pure returns (address token0, addres
    require(tokenA != tokenB, 'UniswapV2Library: IDENTICAL_ADDRESSES');
    (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
    require(token0 != address(0), 'UniswapV2Library: ZERO_ADDRESS');
}

// calculates the CREATE2 address for a pair without making any external calls
function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address
    (address token0, address token1) = sortTokens(tokenA, tokenB);
    pair = address(uint(keccak256(abi.encodePacked(
            hex'ff',
            factory,
            keccak256(abi.encodePacked(token0, token1)),
            PairCodeHash                                   //hex'96e8ac4277198ff8b6f785478aa9a39
        ))));
}

// fetches and sorts the reserves for a pair
function getReserves(address factory, address tokenA, address tokenB) internal view returns (uint
    (address token0,) = sortTokens(tokenA, tokenB);
    (uint reserve0, uint reserve1,) = IUniswapV2Pair(pairFor(factory, tokenA, tokenB)).getReserve
    (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
}

// given some amount of an asset and pair reserves, returns an equivalent amount of the other ass
function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
    require(amountA > 0, 'UniswapV2Library: INSUFFICIENT_AMOUNT');
    require(reserveA > 0 && reserveB > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    amountB = amountA.mul(reserveB) / reserveA;
}

// given an input amount of an asset and pair reserves, returns the maximum output amount of the
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint
    require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint amountInWithFee = amountIn.mul(997);
    uint numerator = amountInWithFee.mul(reserveOut);
    uint denominator = reserveIn.mul(1000).add(amountInWithFee);
    amountOut = numerator / denominator;
}

// given an output amount of an asset and pair reserves, returns a required input amount of the o
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint
    require(amountOut > 0, 'UniswapV2Library: INSUFFICIENT_OUTPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint numerator = reserveIn.mul(amountOut).mul(1000);
    uint denominator = reserveOut.sub(amountOut).mul(997);
    amountIn = (numerator / denominator).add(1);
}

// performs chained getAmountOut calculations on any number of pairs
function getAmountsOut(address factory, uint amountIn, address[] memory path) internal view retur
    require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
    amounts = new uint[](path.length);
    amounts[0] = amountIn;
    for (uint i; i < path.length - 1; i++) {
        (uint reserveIn, uint reserveOut) = getReserves(factory, path[i], path[i + 1]);
        amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
    }
}

// performs chained getAmountIn calculations on any number of pairs
function getAmountsIn(address factory, uint amountOut, address[] memory path) internal view retur
    require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
```

```
        amounts = new uint[](path.length);
        amounts[amounts.length - 1] = amountOut;
        for (uint i = path.length - 1; i > 0; i--) {
            (uint reserveIn, uint reserveOut) = getReserves(factory, path[i - 1], path[i]);
            amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
        }
    }
}

// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: APPROVE_F
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call.value(value)(new bytes(0));        // (bool success,) = to.call{
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
    }
}


contract DeployFactory {
    event Deploy(string name, address addr);

    constructor(address adminFactory, address adminPair) public {
        AiSwapPair pair = new AiSwapPair();
        emit Deploy('AiSwapPair', address(pair));

        AiSwapFactory factory = new AiSwapFactory();
        emit Deploy('AiSwapFactory', address(factory));

        InitializableAdminUpgradeabilityProxy factoryProxy = new InitializableAdminUpgradeabilityProx
        factoryProxy.initialize(adminFactory, address(factory), abi.encodeWithSignature('initialize(a
        emit Deploy('factoryProxy', address(factoryProxy));

        //selfdestruct(msg.sender);
    }
}

contract DeployRouter {
    event Deploy(bytes32 name, address addr);

    constructor(address adminRouter, address factoryProxy, address WETH) public {
        if(WETH == address(0))
            WETH = AddressWETH.WETH();
        require(WETH != address(0), 'AiSwapFactoryFactory: WETH address is 0x0');

        AiSwapRouter02 router = new AiSwapRouter02();
        router.initialize(address(factoryProxy), WETH);
        emit Deploy('AiSwapRouter02', address(router));
        InitializableAdminUpgradeabilityProxy routerProxy = new InitializableAdminUpgradeabilityProxy
        routerProxy.initialize(adminRouter, address(router), abi.encodeWithSignature('initialize(addr
        emit Deploy('routerProxy', address(routerProxy));

        //selfdestruct(msg.sender);
    }
}

contract Test {
    function pairFor(address factory, address tokenA, address tokenB) public pure returns (address) {
        return UniswapV2Library.pairFor(factory, tokenA, tokenB);
    }

    function pairCreationCode() public pure returns (bytes memory) {
        return type(InitializableProductProxy).creationCode;
    }

    function pairCodeHash() public pure returns (bytes32) {
        return UniswapV2Library.pairCodeHash();
    }

    function pairCodeHash2() public pure returns (bytes32) {
        return keccak256(type(InitializableProductProxy).creationCode);
    }
}

library AddressWETH {
    function WETH() internal pure returns (address addr) {
        assembly {
            switch chainid()
                case  1  { addr := 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 }      // Ethereum Main
                case  3  { addr := 0xc778417E063141139Fce010982780140Aa0cD5Ab }      // Ethereum Test
                case  4  { addr := 0xc778417E063141139Fce010982780140Aa0cD5Ab }      // Ethereum Test
                case  5  { addr := 0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6 }      // Ethereum Test
                case 42  { addr := 0xd0A1E359811322d97991E03f863a0C30C2cF029C }      // Ethereum Test
                case 56  { addr := 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c }      // BSC Mainnet
                case 65  { addr := 0x2219845942d28716c0f7c605765fabdca1a7d9e0 }      // OKExChain Tes
                case 66  { addr := 0x8f8526dbfd6e38e3d8307702ca8469bae6c56c15 }      // OKExChain Mai
                case 128 { addr := 0x5545153ccfca01fbd7dd11c0b23ba694d9509a6f }      // HECO Mainnet
                case 256 { addr := 0xB49f19289857f4499781AaB9afd4A428C4BE9CA8 }      // HECO Testnet
                default  { addr := 0x0                                        }      // unknown
        }
    }
}
```

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 specifers 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!
```

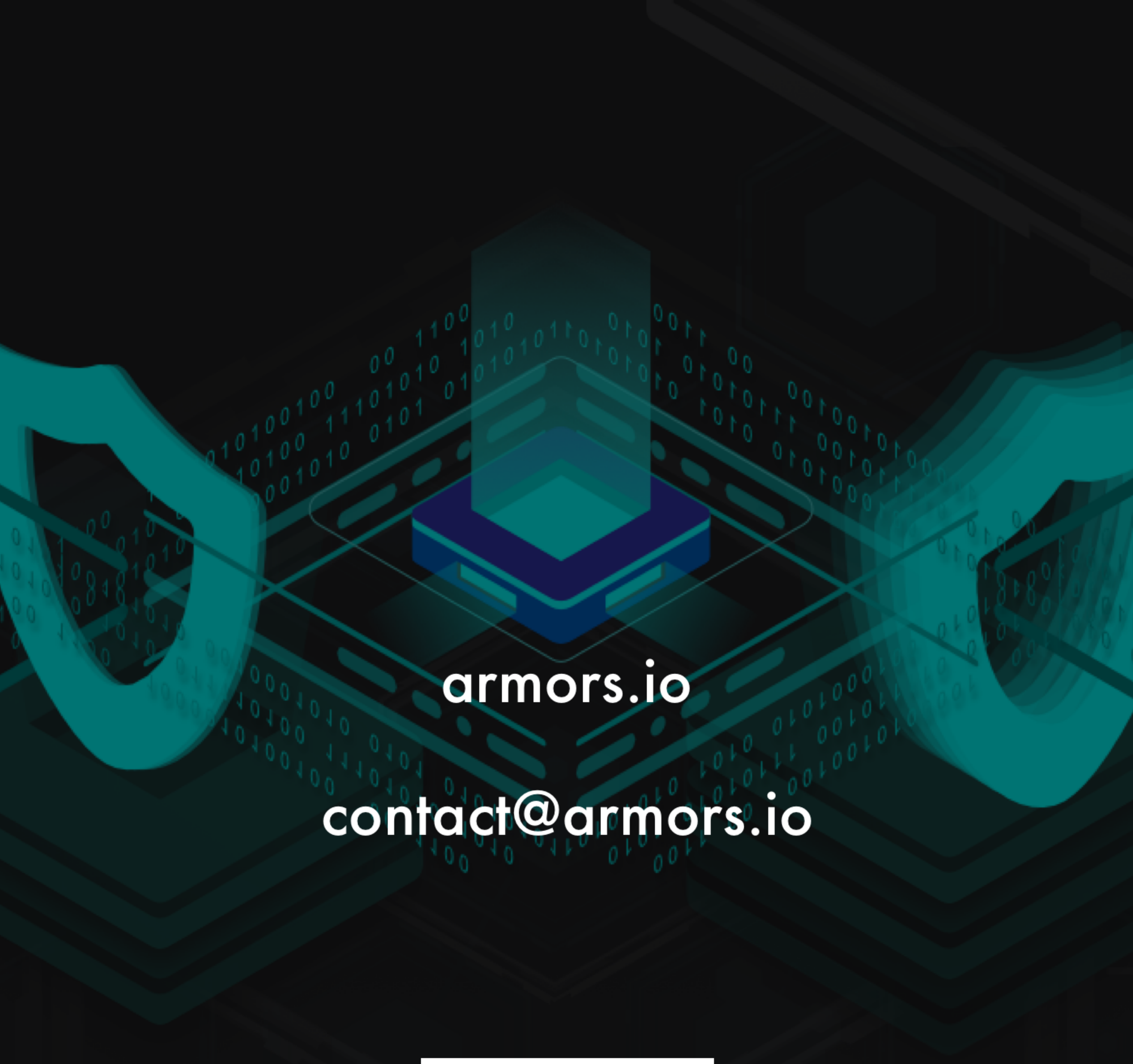
- **Security suggestion:**
  no.

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