KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: Knownsec team for blockchain security

Version

Content	Date	Revisor	Version
document	20201012	Knownsec team for blockchain security	V1.0

Details

Name	Version	Code	Confidentiality level
IDMO smart contract audit	V1.0	IDMO-ZNNY-20201012	open

Statement

Knownsec only issues the report and assumes responsibilities for what have occurred or existed before. Knownsec does not take responsibilities for what will happen in the future in that the security of smart contracts has not been ensured. The security audit analysis in this report is only based on the documents and information provided to Knownsec before this report is issued. Knownsec believes that there is no information missing, tampered, deleted or concealed in the document provided. If so, Knownsec will not bear any responsibility for the losses and adverse effects caused thereby.

Content

1.Summary

Valid testing was carried out from October 10, 2020 to October 12, 2020. During this period, the security and standardization of IDMO smart contract code was audited on which the statistics report is based.

In this test, Knownsec engineers conducted a comprehensive analysis of the common vulnerabilities in smart contracts (see Chapter 3), and the smart contract has passed the comprehensive assessment.

Result of this smart contract security audit: pass

This testing is carried out in a non-production environment, and all codes are up-to-date, and we have communicated with the relevant interface person about the testing which is carried out under the controllable environment to avoid the operational risk and code security risk during the testing.

Subject in this testing

Item	Description	
Token name	IDMO	
Contract address	MyToken	0x4Ba376dec87EDaa662Cd82278d8940 6864118EFd
	IDMO	0x8D63A7416466832AAaB1482E4225 0F5D05B309B8
Code type	token code, DeFi protocol code and Ethereum smart contract code	
Code language	Solidity	

Contract document and MD5

Contract document	MD5
IDMOToken.sol	71df848c2445f4dc19e7ffb54ed266fb

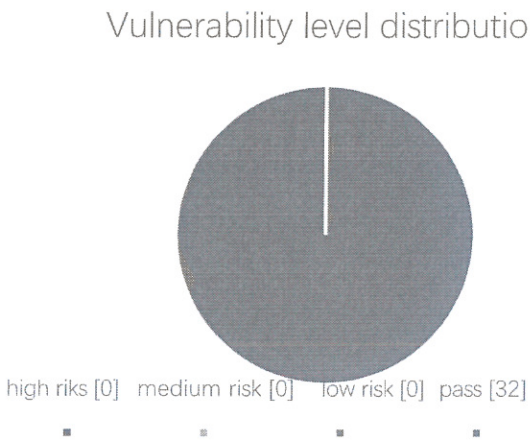
IDMO.sol	c48b2c76f2c15680b09b4abc029e3da2

2.Code vulnerability analysis

2.1 Vulnerability level distribution

Vulnerability data on different risk levels:

Statistics on security risk levels			
High risk	Medium risk	Low risk	Pass
0	0	0	32



2.2 Summary of audit results

Audit results			
Item	**Content**	**State**	**Details**
Function security check	MyToken contract token	pass	There are no safety vulnerabilities found in testing.
	IDMO contract token-adding function	pass	There are no safety vulnerabilities found in testing.
	IDMO contract mining-adding function	pass	There are no safety vulnerabilities found in testing.
	IDMO contract deposit function	pass	There are no safety vulnerabilities found in testing.
	IDMO contract withdraw function	pass	There are no safety vulnerabilities found in testing.
Code vulnerabilities testing	Compiler version security	pass	There are no safety vulnerabilities found in testing.
	Redundant code	pass	There are no safety vulnerabilities found in testing.
	Use of safe arithmetic library	pass	There are no safety vulnerabilities found in testing.
	Encoding not recommended	pass	There are no safety vulnerabilities found in testing.
	Rational use of require/assert	pass	There are no safety vulnerabilities found in testing.
	Fallback function security	pass	There are no safety vulnerabilities found in testing.
	Authorization through tx.orgin	pass	There are no safety vulnerabilities found in testing.
	Owner authority control	pass	There are no safety vulnerabilities found in testing.
	Gas consumption testing	pass	There are no safety vulnerabilities found in testing.
	Call injection attack	pass	There are no safety vulnerabilities found in testing.
	Low-level function safety	pass	There are no safety vulnerabilities found in testing.
	Vulnerability in increasing token issuance	pass	There are no safety vulnerabilities found in testing.
	Access control defect detection	pass	There are no safety vulnerabilities found in testing.

	Integer overflow and underflow detection	pass	There are no safety vulnerabilities found in testing.
	Arithmetic accuracy error	pass	There are no safety vulnerabilities found in testing.
	Weak sources of randomness	pass	There are no safety vulnerabilities found in testing.
	Use of unsafe interface	pass	There are no safety vulnerabilities found in testing.
	Variable coverage	pass	There are no safety vulnerabilities found in testing.
	Uninitialized storage pointer	pass	There are no safety vulnerabilities found in testing.
	Call return value verification	pass	There are no safety vulnerabilities found in testing.
	Transaction order dependence verification	pass	There are no safety vulnerabilities found in testing.
	Timestamp dependence attack	pass	There are no safety vulnerabilities found in testing.
	Denial of service attack detection	pass	There are no safety vulnerabilities found in testing.
	False top-up vulnerability detection	pass	There are no safety vulnerabilities found in testing.
	Re-entrancy attack detection	pass	There are no safety vulnerabilities found in testing.
	Replay attack detection	pass	There are no safety vulnerabilities found in testing.
	Reordering attack detection	pass	There are no safety vulnerabilities found in testing.

3.3.Security Testing

3.1 MyToken contract token【pass】

Audit analysis: MyToken token contract is reasonably designed and meets ERC20 standards.

```
contract ERC20 is Context, IERC20 {//knownsec// ERC20 standard
    using SafeMath for uint256;
    using Address for address;
    mapping (address => uint256) private _balances;
    mapping (address => mapping (address => uint256)) private _allowances;
    uint256 private _totalSupply;
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
        _decimals = 18;
    }
    function name() public view returns (string memory) {
        return _name;
    }
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    function decimals() public view returns (uint8) {
        return _decimals;
    }
    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }
   function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
```

```
    }
    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true;
    }
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
        _beforeTokenTransfer(sender, recipient, amount);
        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");
        _beforeTokenTransfer(address(0), account, amount);
        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

```
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");
        _beforeTokenTransfer(account, address(0), amount);
        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}
contract MyToken is ERC20("IDMOToken", "IDMO"), Ownable {
    function mint(address _to, uint256 _amount) public onlyOwner {/// knownsec// add tokens only
available for owner
        _mint(_to, _amount);
    }
```

Security advice： none.

3.2 IDMO contract token-adding function【pass】

Audit analysis: AddToken function is used to add tokens in smart contract, in which checkTokenParam function is used to verify the token parameters input.

```
function checkTokenParam(TokenParam memory tokenParam) public view{
        require(tokenParam.myTokenAddr != address(0), "myTokenAddr error");//knownsec// check token address
        isTokenExist(tokenParam.myTokenAddr);//knownsec// check token unaddedcheck token unadded
        require(tokenParam.devAddr != address(0), "devAddr error");//knownsec// check dev address
        require(((tokenParam.amount1st>0 && tokenParam.blkNum1st>=10000) || (tokenParam.amount1st==0 && tokenParam.blkNum1st==0)), "amount1st blkNum1st error");
        require(((tokenParam.amount2nd>0 && tokenParam.blkNum2nd>=10000) || (tokenParam.amount2nd==0 && tokenParam.blkNum2nd==0)), "amount2nd blkNum2nd error");
        require(((tokenParam.amount3rd>0 && tokenParam.blkNum3rd>=10000) || (tokenParam.amount3rd==0 && tokenParam.blkNum3rd==0)), "amount3rd blkNum3rd error");
        require((tokenParam.feeRate>0 && tokenParam.feeRate<=20), "feeRate error");//knownsec// 0<feeRate<=20
        require(tokenParam.blkNumPriMine >= 10000, "blkNumPriMine error");
}
function addToken(TokenParam memory tokenParam) public onlyControl returns(uint256){//knownsec// add token, only available to owner or contract address
        checkTokenParam(tokenParam);//knownsec// check token parameter
        tokenInfo[tokenIndex] = tokenParam;//knownsec// add tokenIndex into tokenInfo
        uint tokenId = tokenIndex;//knownsec// current tokenIndex is tokenId
        tokenIndex = tokenIndex + 1;//knownsec// tokenIndex accumulation 1
        mapTokenExist[tokenParam.myTokenAddr] = 1;//knownsec// token has been added   return tokenId;
}
```

Security advice： none

3.3 IDMO contract mining-adding function【pass】

Audit analysis: The function of adding a mining pool to the IDMO contract is mainly implemented by the addPool function, and only contract owner or the delegated contract address can write to related information. The same LP Token cannot be added repeatedly.

```
function addPool(uint tokenId, uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyControl {//knownsec// add mining pool,only owner or delegated contract address can write to
```

```
    if (_withUpdate) {
        massUpdatePools(tokenId);//knownsec// renew mining pool
    }
    uint256 lastRewardBlock = block.number > startBlock[tokenId] ? block.number : startBlock[tokenId];
    totalAllocPoint[tokenId] = totalAllocPoint[tokenId].add(_allocPoint);
    poolInfo[tokenId][poolNum[tokenId]] = PoolInfo({//knownsec// add  new mining pool
        lpToken: _lpToken,
                    amount: 0,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accPerShare: 0
    });
            poolNum[tokenId] = poolNum[tokenId].add(1);
}
```

Security advice：none

3.4 IDMO contract deposit function【pass】

Audit analysis: The despoit function is used to deposit mobile tokens to obtain profits.

```
function deposit(uint tokenId, uint256 _pid, uint256 _amount) public {//knownsec// deposit lp token
            if(tokenId != 0){
                    require(startBlock[tokenId] != 0);//knownsec// check token and mobile mining is available
                    require(tokenInfo[tokenId].blkNumPriMine + startBlock[tokenId] <= block.number, "priority period");
            }
    PoolInfo storage pool = poolInfo[tokenId][_pid];
    UserInfo storage user = userInfo[tokenId][_pid][msg.sender];
    updatePool(tokenId, _pid);

    if (user.amount > 0) {
        uint256 pending = user.amount.mul(pool.accPerShare).div(1e12).sub(user.rewardDebt);
        safeTokenTransfer(tokenId, msg.sender, pending);
    }
    pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);//knownsec// Transfer mobile token lpToken
    user.amount = user.amount.add(_amount);
            pool.amount = pool.amount.add(_amount);
```

```
    user.rewardDebt = user.amount.mul(pool.accPerShare).div(1e12);
    emit Deposit(msg.sender,tokenId, _pid, _amount);
}
```

Security advice： none

3.5 IDMO contract withdraw function【pass】

Audit analysis: The withdraw function is used to withdraw the profits in mobile mining.

```
function withdraw(uint tokenId, uint256 _pid, uint256 _amount) public {//knownsec// lp token withdrawal
    PoolInfo storage pool = poolInfo[tokenId][_pid];
    UserInfo storage user = userInfo[tokenId][_pid][msg.sender];
    require(user.amount >= _amount, "withdraw: not good");//knownsec// check balance
        if(user.lock_expire != 0){//knownsec// check user freeze
            if(user.lock_expire > now){
                require(user.amount.sub(user.lock_amount) >= _amount,"lock amount");
            }
            else{
                user.lock_expire = 0;
                user.lock_amount = 0;
            }
    }
    updatePool(tokenId, _pid);
    uint256 pending = user.amount.mul(pool.accPerShare).div(1e12).sub(user.rewardDebt);
    safeTokenTransfer(tokenId, msg.sender, pending);
    user.amount = user.amount.sub(_amount);
    pool.amount = pool.amount.sub(_amount);
    user.rewardDebt = user.amount.mul(pool.accPerShare).div(1e12);
    pool.lpToken.safeTransfer(address(msg.sender), _amount);//knownsec// withdrawal
    emit Withdraw(msg.sender,tokenId, _pid, _amount);
}
```

Security advice： none.

4.Code vulnerabilities testing

4.1 Compiler version security【pass】

Check whether a safe version of compiler is used in the contract code

Test result: After testing, the compiler version formulated is 0.6.12 or above in the smart contract code, and there is no such security issue.

Security advice: none.

4.2 Redundant code【pass】

Check whether there are redundant codes in contract code

Test result: After testing, the security problem does not exist in the smart contract code.

Security advice: none.

4.3 Use of safe arithmetic library【pass】

Check whether the SafeMath safe arithmetic library is used in the contract code

Test result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no security problem.

Security advice: none.

4.4 Encoding not recommended【pass】

Check whether there is an encoding without official recommendation or abandoned in the contract code

Test result: After testing, the security problem does not exist in the smart contract code

Security advice: none.

4.5 Rational use of require/assert【pass】

Check the rationality of require/assert use in the contract code

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none

4.6 Fallback function security【pass】

Check whether the fallback function is used correctly in the contract code

Test result: testing, there is no such security problem in the smart contract code.

Security advice: none.

4.7 Authentication through tx.origin【pass】

tx.origin is a global variable of Solidity that runs through the entire call stack and returns to the address of the account that originally used to send the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks such as phishing.

Test result: testing, there is no such security problem in the smart contract code.

Security advice: none.

4.8 Owner authority control【pass】

Check whether owners in the contract code has excessive authority. For example, check whether they can arbitrarily modify balances in other accounts.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.9 Gas consumption testing【pass】

Check whether gas consumption exceeds the maximum in block

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.10 Call injection attack【pass】

Authority should be strictly controlled when the call function is used, or the function should be set fixed.

Test result: After testing, call function is not used, and there is no such security problem in the smart contract code.

Security advice: none.

4.11 Low-level function safety【pass】

Check whether there are code loopholes in the use of low-level functions (call/delegatecall) during the contract code implementation. The execution of the call function is in the called contract; while the delegate/call function is executed in the current contract of the function

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none

4.12 Vulnerability in increasing token issuance【pass】

Check whether there is a function that may increase the total amount of tokens in the contract after initializing it

Test result: After testing, the smart contract code contains the function of issuing additional tokens. The owner of the token contract can arbitrarily issue additional tokens, but it is found that the owner's authority has been transferred to the mining contract IDMO. Issuing more tokens is required for mobile mining, and additional tokens will only be issued to distribute rewards when new mining pool is added, so the function has passed the testing.

Security advice: none.

4.13 Access control defect detection【pass】

Different functions in the contract should set reasonable authorities.

Check whether each function in the contract correctly uses keywords such as public, private for visibility modification, and check whether the contract is correctly defined and used modifier to restrict key functions to avoid problems caused by unauthorized access.

Test result: After testing, the security problem does not exist in the smart contract code.

Security advice: none.

4.14 Integer overflow and underflow detection【pass】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can process numbers up to 256-bit (2^256-1). If the maximum increases by 1, it will overflow to 0. Similarly, if the number is unsigned, it will underflow to the maximum after you subtract 1 from 0.

Integer overflow and underflow are not new, but they can be highly risky especially in smart contracts. Overflow can lead to incorrect results, and it may decrease the reliability and safety of the program especially when the scenario is not expected.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.15 Arithmetic accuracy error【pass】

Similar to other programming languages, Solidity is equipped with data structure designs, such as variable, constant, function, array and struct. There is, however, a big difference between Solidity and other programming languages. There are no floating-points in Solidity. All results of Solidity will only be integers, and it cannot be used to define decimal type. Calculations are indispensable in the contract, and the design of it may cause relative errors. For example, 5/2*10=20, while 5*10/2=25, which results in more serious and obvious errors when it comes to bigger data.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.16 Weak sources of randomness【pass】

Random numbers may be used in smart contract. Although the functions in Solidity can access unpredictable values, such as block.number and block.timestamp, they are usually more public than expected and can be affected by miners, which means these random numbers are predictable to a certain extent, so malicious parties can usually copy it and attack the function by exploiting its predictability .

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.17 Use of unsafe interface【pass】

Check whether unsafe interface is used in contract code.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.18 Variable coverage【pass】

Check whether there are vulnerabilities caused by variable coverage in contract code

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.19 Uninitialized storage pointer【pass】

There is a special data structure, struct, in solidity, and storage and memory are used in the default settings of local variables.

Being stored in storage (memory) differs from that of memory (inner storage). Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will lead to the fact that variables may point to other storage variables, leading to variable coverage, or even more serious issues.

As a consequence, initializing struct variables should be avoided in the development of function.

Test result: After testing, the smart contract code does not use structure, and there is no such problem in smart contract code.

Security advice: none.

4.20 Call return value verification【pass】

This problem is common in smart-contract-related smart contracts, so it is also called silent failure or unchecked delivery.

There are transfer(), send(), call.value() and other methods of currency transfer in Solidity, which can all be used to send Ether to a certain address. The difference among them is: When transaction fails in transfer, it will throw and the state will be rolled back. Only 2300gas will be available to prevent reentry attacks, while transaction will be returned to

false when trade fails in send. Only 2300gas will be available to prevent reentry attacks, and transaction will be returned to false when it fails in call.value.

All available gas will be used (The figure can be set by importing gas_value parameters), however it cannot effectively prevent re-entrancy attacks.

If the return value above in currency transfer functions of send and call.value is not checked in the code, the following code will continue to be executed in the contract, which may lead to unexpected results due to Ether which is not successfully sent.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.21 Transaction order dependence verification【pass】

Miners gain gas fees through EOA, thus users can specify higher gas fees for faster transactions. The Ethereum blockchain is public, and everyone can access transactions of others. This means that if a user comes up with a viable solution, a malicious party can steal and copy it for a higher fee, covering the original solution.

Test result: After testing, there is no such security problem in the smart contract code.

```
function deposit() public {//knownsec// mobile mining
    uint _want = IERC20(want).balanceOf(address(this));
    address _controller = For(fortube).controller();
    if (_want > 0) {
        //knownsec// authority data cannot be 0 in HBTC contract
        // IERC20(want).safeApprove(_controller, 0);
        IERC20(want).safeApprove(_controller, _want);
        For(fortube).deposit(want,_want);
    }
}
```

Security advice: none.

4.22 Timestamp dependency attack【pass】

The timestamp in data block is usually consistent with the miner's local time, and error margin is 900 seconds. When a new block is accepted by other nodes, they only need to verify whether the timestamp is later than that in the previous block, and they should make sure that the deviation from the local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy all the requirements beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.23 Denial of service attack【pass】

In Ethereum, denial of service is fatal, and a smart contract attacked by it may never be able to return to normal. There are many reasons for the denial of service in smart contract. For example, gas increased by malicious parties in computing may cause gas exhaustion. Access control aimed at accessing private components of the smart contract can also be abused. In addition, confusing information and negligence may be exploited.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.24 False top-up vulnerability detection【pass】

If function is used to check balances of the transfer initiator (msg.sender) in the transfer function of token contract. When balances[msg.sender] <value, the statements inside the body of else will be executed and return false, and finally no exception will be thrown in the end. We believe that it is not prudent to execute if/else, a lenient function in sensitivity function when it comes to transfer.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.25 Re-entrancy attack detection【pass】

Re-entrancy attack is the most famous vulnerability in Ethereum smart contract, which once led to the fork of Ethereum (The DAO hack).

All the gas fee received will be consumed when Ether is sent via call.value() function in Solidit. The contract is vulnerable to re-entrancy attack if Ether is sent before the balance of user is decreased.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

4.26 Replay attack detection【pass】

If entrusted management is needed in contract, then the non-reusability should be verified to avoid replay attacks. In asset management system, entrusted management is ubiquitous in which the trustee pays a certain fee to the trustee who can manages assets. This business is also common in smart contracts.

Test result: After detection, the smart contract does not use the call function, and this vulnerability does not exist.

Security advice: none.

4.27 Reordering attack detection【pass】

In a reordering attack, miners or others try to compete with other participants in smart contract by inserting their own information into a list or mapping, so that they have the opportunity to store their own information into contract.

Test result: After testing, there is no such security problem in the smart contract code.

Security advice: none.

5.Appendix A:Contract code

Code resource of the testing:

IDMOToken.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-23
*/

pragma solidity 0.6.12;

  //https://www.idmoswap.com

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

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

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

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


  function approve(address spender, uint256 amount) external returns (bool);
```

```
    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

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

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


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

        return c;
    }

    /**
```

```
    * @dev Returns the subtraction of two unsigned integers, reverting on
    * overflow (when the result is negative).
    *
    * Counterpart to Solidity's `-` operator.
    *
    * Requirements:
    *
    * - Subtraction cannot overflow.
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
    * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
    * overflow (when the result is negative).
    *
    * Counterpart to Solidity's `-` operator.
    *
    * Requirements:
    *
    * - Subtraction cannot overflow.
    */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
    * @dev Returns the multiplication of two unsigned integers, reverting on
    * overflow.
    *
    * Counterpart to Solidity's `*` operator.
    *
    * Requirements:
    *
    * - Multiplication cannot overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {

        if (a == 0) {
            return 0;
        }
```

```
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

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

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
    *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
```

```
    * Counterpart to Solidity's `%` operator. This function uses a `revert`
    * opcode (which leaves remaining gas untouched) while Solidity uses an
    * invalid opcode to revert (consuming all remaining gas).
    *
    * Requirements:
    *
    * - The divisor cannot be zero.
    */
   function mod(uint256 a, uint256 b) internal pure returns (uint256) {
       return mod(a, b, "SafeMath: modulo by zero");
   }


   function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
       require(b != 0, errorMessage);
       return a % b;
   }
}


library Address {

   function isContract(address account) internal view returns (bool) {
       // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
       // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
       // for accounts without code, i.e. `keccak256('')`
       bytes32 codehash;
       bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
       // solhint-disable-next-line no-inline-assembly
       assembly { codehash := extcodehash(account) }
       return (codehash != accountHash && codehash != 0x0);
   }


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

       // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
       (bool success, ) = recipient.call{ value: amount }("");
       require(success, "Address: unable to send value, recipient may have reverted");
   }


   function functionCall(address target, bytes memory data) internal returns (bytes memory) {
     return functionCall(target, data, "Address: low-level call failed");
   }
```

```
    function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory)
{
        return _functionCallWithValue(target, data, 0, errorMessage);
    }


    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }


    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage)
internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

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

        // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

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


library SafeERC20 {
```

```
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }


    function safeApprove(IERC20 token, address spender, uint256 value) internal {

        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased allowance
below zero");
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }


    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which verifies that
        // the target address contains contract code and also asserts for success in the low-level call.

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
```

```
library EnumerableSet {


  struct Set {
    // Storage of set values
    bytes32[] _values;

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


  function _add(Set storage set, bytes32 value) private returns (bool) {
    if (!_contains(set, value)) {
      set._values.push(value);
      // The value is stored at length-1, but we add 1 to all indexes
      // and use 0 as a sentinel value
      set._indexes[value] = set._values.length;
      return true;
    } else {
      return false;
    }
  }

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

    if (valueIndex != 0) { // Equivalent to contains(set, value)
      // To delete an element from the _values array in O(1), we swap the element to delete with the last one in
      // the array, and then remove the last element (sometimes called as 'swap and pop').
      // This modifies the order of the array, as noted in {at}.

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

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

```
        bytes32 lastvalue = set._values[lastIndex];

        // Move the last value to the index where the value to delete is
        set._values[toDeleteIndex] = lastvalue;
        // Update the index for the moved value
        set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

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

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

        return true;
    } else {
        return false;
    }
}


function _contains(Set storage set, bytes32 value) private view returns (bool) {
    return set._indexes[value] != 0;
}


function _length(Set storage set) private view returns (uint256) {
    return set._values.length;
}


function _at(Set storage set, uint256 index) private view returns (bytes32) {
    require(set._values.length > index, "EnumerableSet: index out of bounds");
    return set._values[index];
}


struct AddressSet {
    Set _inner;
}


function add(AddressSet storage set, address value) internal returns (bool) {
    return _add(set._inner, bytes32(uint256(value)));
}
```

```
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(value)));
    }


    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(value)));
    }

    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }


    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint256(_at(set._inner, index)));
    }



    struct UintSet {
        Set _inner;
    }


    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }


    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }


    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }


    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
```

```
        return uint256(_at(set._inner, index));
    }
}


abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

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


contract Ownable is Context {
    address private _owner;

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


    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }


    function owner() public view returns (address) {
        return _owner;
    }


    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }


    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
```

```
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}


contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;


    constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
        _decimals = 18;
    }


    function name() public view returns (string memory) {
        return _name;
    }


    function symbol() public view returns (string memory) {
        return _symbol;
    }


    function decimals() public view returns (uint8) {
        return _decimals;
    }
```

```
    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }


    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }


    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }


    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds
allowance"));
        return true;
    }


    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }


    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased
allowance below zero"));
        return true;
    }

function _transfer(address sender, address recipient, uint256 amount) internal virtual {
```

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

    _beforeTokenTransfer(sender, recipient, amount);

    _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}


function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

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

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}


function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

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

    _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}


function _approve(address owner, address spender, uint256 amount) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

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


function _setupDecimals(uint8 decimals_) internal {
    _decimals = decimals_;
}
function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
```

```
}


contract MyToken is ERC20("IDMOToken", "IDMO"), Ownable {
    function mint(address _to, uint256 _amount) public onlyOwner {//knownsec// add token and only owner can write to
        _mint(_to, _amount);
    }
}
pragma experimental ABIEncoderV2;
```

IDMO.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-30
*/

// https://idmoswap.com/
pragma solidity 0.6.12;

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

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
```

```
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

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

/**
 * @dev Moves `amount` tokens from `sender` to `recipient` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

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

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

```
}



/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

    return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
```

```
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a, b, "SafeMath: subtraction overflow");
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 *
 * - Subtraction cannot overflow.
 */
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b <= a, errorMessage);
    uint256 c = a - b;

    return c;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `*` operator.
 *
 * Requirements:
 *
 * - Multiplication cannot overflow.
 */
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
    if (a == 0) {
        return 0;
    }

    uint256 c = a * b;
    require(c / a == b, "SafeMath: multiplication overflow");

    return c;
}

/**
```

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

/**
 * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses a
 * `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b > 0, errorMessage);
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold

    return c;
}

/**
 * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
 * Reverts when dividing by zero.
 *
 * Counterpart to Solidity's `%` operator. This function uses a `revert`
 * opcode (which leaves remaining gas untouched) while Solidity uses an
 * invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
```

```
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

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



/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
```

```
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

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

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-
```

```
decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory)
{
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage)
```

```
internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

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

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

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


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
```

```
    _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
  }

  function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
    _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
  }

  /**
   * @dev Deprecated. This function has issues similar to the ones found in
   * {IERC20-approve}, and its usage is discouraged.
   *
   * Whenever possible, use {safeIncreaseAllowance} and
   * {safeDecreaseAllowance} instead.
   */
  function safeApprove(IERC20 token, address spender, uint256 value) internal {
    // safeApprove should only be called when setting an initial allowance,
    // or when resetting it to zero. To increase and decrease it, use
    // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
    // solhint-disable-next-line max-line-length
    require((value == 0) || (token.allowance(address(this), spender) == 0),
      "SafeERC20: approve from non-zero to non-zero allowance"
    );
    _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
  }

  function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
    uint256 newAllowance = token.allowance(address(this), spender).add(value);
    _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
  }

  function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
    uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased allowance
below zero");
    _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
  }

  /**
   * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
   * on the return value: the return value is optional (but if data is returned, it must not be false).
   * @param token The token targeted by the call.
   * @param data The call data (encoded using abi.encode or one of its variants).
   */
  function _callOptionalReturn(IERC20 token, bytes memory data) private {
    // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
    // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which verifies that
    // the target address contains contract code and also asserts for success in the low-level call.
```

```
        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 * ```
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 * ```
 *
 * As of v3.0.0, only sets of type `address` (`AddressSet`) and `uint256`
 * (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;
```

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

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

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

    if (valueIndex != 0) { // Equivalent to contains(set, value)
        // To delete an element from the _values array in O(1), we swap the element to delete with the last one in
        // the array, and then remove the last element (sometimes called as 'swap and pop').
        // This modifies the order of the array, as noted in {at}.

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

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

        bytes32 lastvalue = set._values[lastIndex];
```

```
        // Move the last value to the index where the value to delete is
        set._values[toDeleteIndex] = lastvalue;
        // Update the index for the moved value
        set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

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

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

        return true;
    } else {
        return false;
    }
}

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

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

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

```
// AddressSet

struct AddressSet {
    Set _inner;
}

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

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

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

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

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

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


// UintSet

struct UintSet {
    Set _inner;
}

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

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

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

/**
 * @dev Returns the number of values on the set. O(1).
 */
```

```
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

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



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

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}
```

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

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

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
```

```
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}



/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
```

```
using SafeMath for uint256;
using Address for address;

mapping (address => uint256) private _balances;

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

uint256 private _totalSupply;

string private _name;
string private _symbol;
uint8 private _decimals;

/**
 * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
 * a default value of 18.
 *
 * To select a different value for {decimals}, use {_setupDecimals}.
 *
 * All three of these values are immutable: they can only be set once during
 * construction.
 */
constructor (string memory name, string memory symbol) public {
    _name = name;
    _symbol = symbol;
    _decimals = 18;
}

/**
 * @dev Returns the name of the token.
 */
function name() public view returns (string memory) {
    return _name;
}

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

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
```

```
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }

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

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

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

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

```
    }

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

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds
allowance"));
        return true;
    }

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

```
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
    */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
```

```
    emit Transfer(sender, recipient, amount);
}

/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements
 *
 * - `to` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

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

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}

/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

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

    _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
 *
```

```
    * This is internal function is equivalent to `approve`, and can be used to
    * e.g. set automatic allowances for certain subsystems, etc.
    *
    * Emits an {Approval} event.
    *
    * Requirements:
    *
    * - `owner` cannot be the zero address.
    * - `spender` cannot be the zero address.
    */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
    * @dev Sets {decimals} to a value other than the default one of 18.
    *
    * WARNING: This function should only be called from the constructor. Most
    * applications that interact with token contracts will not expect
    * {decimals} to ever change, and may work incorrectly if it does.
    */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    /**
    * @dev Hook that is called before any transfer of tokens. This includes
    * minting and burning.
    *
    * Calling conditions:
    *
    * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
    * will be to transferred to `to`.
    * - when `from` is zero, `amount` tokens will be minted for `to`.
    * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
    * - `from` and `to` are never both zero.
    *
    * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
    */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}
```

```
contract MyToken is ERC20("IDMOToken", "IDMO"), Ownable {

    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
    }
}



pragma experimental ABIEncoderV2;

contract IDMO is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;
        uint256 rewardDebt;
                        uint256 lock_expire;
                        uint256 lock_amount;
    }

    // Info of each pool.
    struct PoolInfo {
        IERC20 lpToken;
                        uint256 amount;
        uint256 allocPoint;
        uint256 lastRewardBlock;
        uint256 accPerShare;
    }

            struct TokenParam{
                        address myTokenAddr;
                        address devAddr;
                        uint amount1st;
                        uint blkNum1st;
                        uint amount2nd;
                        uint blkNum2nd;
                        uint amount3rd;
                        uint blkNum3rd;
                        uint feeRate;
                        uint blkNumPriMine;
```

```
    }

    mapping (uint256 => mapping(uint256 => PoolInfo)) public poolInfo;

    mapping (uint256 => mapping(uint256 => mapping (address => UserInfo))) public userInfo;

mapping(uint256=>uint256) public totalAllocPoint;


    mapping(uint256=>uint256) public startBlock;

    uint256 public tokenIndex; //knownsec// token index search token in token Info mapping


    mapping(uint256=>TokenParam) public tokenInfo;//knownsec// store token parameters with tokenIndex indexed

    mapping(uint256=>uint256) public poolNum;//knownsec// token and mining pool number

    address public delegateContract;

    mapping(address=>uint256) public mapTokenExist;//knownsec// map added token

    event Deposit(address indexed user, uint256 indexed tokenid, uint256 indexed pid, uint256 amount);
event Withdraw(address indexed user, uint256 indexed tokenid, uint256 indexed pid, uint256 amount);
event EmergencyWithdraw(address indexed user, uint256 indexed tokenid, uint256 indexed pid, uint256 amount);

    modifier onlyControl(){
        address contractOwner = owner();
        require((msg.sender == contractOwner || msg.sender == delegateContract), "Caller error.");
        _;
    }
    modifier onlyDelegate(){
        require(msg.sender == delegateContract, "caller error");
        _;
    }


    function setDelegateContract(address _addr) public onlyOwner{

        delegateContract = _addr;
    }

    function isTokenExist(address tokenAddr) public  view{
        require(mapTokenExist[tokenAddr] == 0, "token exists");
    }
```

```
        function checkTokenParam(TokenParam memory tokenParam) public  view{
                require(tokenParam.myTokenAddr  !=  address(0),  "myTokenAddr  error");//knownsec//  verify  token
address
                isTokenExist(tokenParam.myTokenAddr);//knownsec// verify token
                require(tokenParam.devAddr != address(0), "devAddr error");//knownsec// verify dev address
                require(((tokenParam.amount1st>0            &&            tokenParam.blkNum1st>=10000)            ||
(tokenParam.amount1st==0 && tokenParam.blkNum1st==0)), "amount1st blkNum1st error");
                require(((tokenParam.amount2nd>0            &&            tokenParam.blkNum2nd>=10000)            ||
(tokenParam.amount2nd==0 && tokenParam.blkNum2nd==0)), "amount2nd blkNum2nd error");
                require(((tokenParam.amount3rd>0            &&            tokenParam.blkNum3rd>=10000)            ||
(tokenParam.amount3rd==0 && tokenParam.blkNum3rd==0)), "amount3rd blkNum3rd error");
                require((tokenParam.feeRate>0    &&    tokenParam.feeRate<=20),    "feeRate    error");//knownsec//
0<feeRate<=20
                require(tokenParam.blkNumPriMine >= 10000, "blkNumPriMine error");
        }


        constructor(TokenParam memory tokenParam) public {
    checkTokenParam(tokenParam);
                tokenInfo[tokenIndex] = tokenParam;
                tokenIndex = tokenIndex + 1;
                mapTokenExist[tokenParam.myTokenAddr] = 1;
  }

        function addToken(TokenParam memory tokenParam) public onlyControl returns(uint256){//knownsec// add
token and only owner and entrusted contract address can write to
                checkTokenParam(tokenParam);//knownsec// token parameter
                tokenInfo[tokenIndex] = tokenParam;//knownsec// add tokenIndex into tokenInfo
                uint tokenId = tokenIndex;//knownsec// current tokenIndex equals to tokenId
                tokenIndex = tokenIndex + 1;//knownsec// tokenIndex accumulate 1
                mapTokenExist[tokenParam.myTokenAddr] = 1;//knownsec// map added token
                return tokenId;
        }


        function setStartBlock(uint tokenId, uint _startBlk) public onlyControl{//knownsec// set start block and only
owner and entrusted contract address can write to
                require(tokenId < tokenIndex);
                require(startBlock[tokenId] == 0);
                require(_startBlk > block.number);
                startBlock[tokenId] = _startBlk;


                uint256 length = poolNum[tokenId];
    for (uint256 pid = 0; pid < length; ++pid) {
      poolInfo[tokenId][pid].lastRewardBlock = _startBlk;
    }
```

```
        }


        function poolLength(uint tokenId) external view returns (uint256) {
    return poolNum[tokenId];
  }


        function addPool(uint tokenId, uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyControl
{//knownsec// add mining pool and only owner and entrusted contract address can write to
    if (_withUpdate) {
      massUpdatePools(tokenId);//knownsec// renew mining pool
    }
    uint256 lastRewardBlock = block.number > startBlock[tokenId] ? block.number : startBlock[tokenId];
    totalAllocPoint[tokenId] = totalAllocPoint[tokenId].add(_allocPoint);
    poolInfo[tokenId][poolNum[tokenId]] = PoolInfo({//knownsec// add new mining pool
      lpToken: _lpToken,
                        amount: 0,
      allocPoint: _allocPoint,
      lastRewardBlock: lastRewardBlock,
      accPerShare: 0
    });
                poolNum[tokenId] = poolNum[tokenId].add(1);
  }


        function setPool(uint tokenId, uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyControl {
    if (_withUpdate) {
      massUpdatePools(tokenId);
    }
    totalAllocPoint[tokenId] = totalAllocPoint[tokenId].sub(poolInfo[tokenId][_pid].allocPoint).add(_allocPoint);
    poolInfo[tokenId][_pid].allocPoint = _allocPoint;
  }


        function pendingToken(uint tokenId, uint256 _pid, address _user) external view returns (uint256) {//knownsec//
check pending token
    PoolInfo storage pool = poolInfo[tokenId][_pid];
    UserInfo storage user = userInfo[tokenId][_pid][_user];
    uint256 accPerShare = pool.accPerShare;
                if(startBlock[tokenId] == 0){
                        return 0;
                }

                uint256 lpSupply = pool.amount;
    if (block.number > pool.lastRewardBlock && lpSupply != 0) {
```

```
        uint256 multiplier = getMultiplier(tokenId, pool.lastRewardBlock, block.number);
        uint256 tokenReward = multiplier.mul(pool.allocPoint).div(totalAllocPoint[tokenId]);
        accPerShare = accPerShare.add(tokenReward.mul(1e12).div(lpSupply));
    }
    return user.amount.mul(accPerShare).div(1e12).sub(user.rewardDebt);
}


    function massUpdatePools(uint tokenId) public {//knownsec// renew mining pool
    uint256 length = poolNum[tokenId];
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(tokenId, pid);
    }
}


    function updatePool(uint tokenId, uint256 _pid) public {//knownsec// renew mining pool
            require(tokenId < tokenIndex);
    PoolInfo storage pool = poolInfo[tokenId][_pid];
            TokenParam storage pram = tokenInfo[tokenId];
    if (block.number <= pool.lastRewardBlock) {
        return;
    }
            if(startBlock[tokenId] == 0){
                return;
            }

            uint256 lpSupply = pool.amount;//knownsec// mobile mining pool supply
    if (lpSupply == 0) {
        pool.lastRewardBlock = block.number;
        return;
    }
    uint256 multiplier = getMultiplier(tokenId, pool.lastRewardBlock, block.number);
    uint256 tokenReward = multiplier.mul(pool.allocPoint).div(totalAllocPoint[tokenId]);
    MyToken(pram.myTokenAddr).mint(pram.devAddr,     tokenReward.mul(pram.feeRate).div(100));//knownsec//     get
reward *feeRate%
    MyToken(pram.myTokenAddr).mint(address(this), tokenReward);//knownsec// get token reward
  pool.accPerShare = pool.accPerShare.add(tokenReward.mul(1e12).div(lpSupply));
    pool.lastRewardBlock = block.number;
}


    function deposit(uint tokenId, uint256 _pid, uint256 _amount) public {//knownsec// deposit lp token
            if(tokenId != 0){
                require(startBlock[tokenId]  != 0);//knownsec// verify designated token mobile mining is
```

```
accessible
                require(tokenInfo[tokenId].blkNumPriMine + startBlock[tokenId] <= block.number,
"priority period");
            }
        PoolInfo storage pool = poolInfo[tokenId][_pid];
        UserInfo storage user = userInfo[tokenId][_pid][msg.sender];
        updatePool(tokenId, _pid);

        if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accPerShare).div(1e12).sub(user.rewardDebt);
            safeTokenTransfer(tokenId, msg.sender, pending);
        }
        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        user.amount = user.amount.add(_amount);
                pool.amount = pool.amount.add(_amount);
        user.rewardDebt = user.amount.mul(pool.accPerShare).div(1e12);
        emit Deposit(msg.sender,tokenId, _pid, _amount);
    }


        function delegateDeposit(address _user, uint tokenId, uint256 _pid, uint256 _amount, uint256 _lock_expire)
public onlyDelegate{
                PoolInfo storage pool = poolInfo[tokenId][_pid];
                UserInfo storage user = userInfo[tokenId][_pid][_user];
                updatePool(tokenId, _pid);
                if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accPerShare).div(1e12).sub(user.rewardDebt);
            safeTokenTransfer(tokenId, _user, pending);
        }
                pool.lpToken.safeTransferFrom(delegateContract, address(this), _amount);
        user.amount = user.amount.add(_amount);
                user.lock_amount = user.lock_amount.add(_amount);
                user.lock_expire = _lock_expire;
                pool.amount = pool.amount.add(_amount);
        user.rewardDebt = user.amount.mul(pool.accPerShare).div(1e12);
        emit Deposit(_user,tokenId, _pid, _amount);
            }


    function withdraw(uint tokenId, uint256 _pid, uint256 _amount) public {//knownsec// lp token withdrawal
        PoolInfo storage pool = poolInfo[tokenId][_pid];
        UserInfo storage user = userInfo[tokenId][_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
                if(user.lock_expire != 0){
                        if(user.lock_expire > now){
                                require(user.amount.sub(user.lock_amount) >= _amount,"lock amount");
```

```
                }
                else{
                        user.lock_expire = 0;
                        user.lock_amount = 0;
                }
        }
    updatePool(tokenId, _pid);
    uint256 pending = user.amount.mul(pool.accPerShare).div(1e12).sub(user.rewardDebt);
    safeTokenTransfer(tokenId, msg.sender, pending);
    user.amount = user.amount.sub(_amount);
        pool.amount = pool.amount.sub(_amount);
    user.rewardDebt = user.amount.mul(pool.accPerShare).div(1e12);
    pool.lpToken.safeTransfer(address(msg.sender), _amount);
    emit Withdraw(msg.sender,tokenId, _pid, _amount);
}

    function safeTokenTransfer(uint tokenId, address _to, uint256 _amount) internal {
        TokenParam storage pram = tokenInfo[tokenId];
        uint256 Bal = ERC20(pram.myTokenAddr).balanceOf(address(this));
    if (_amount > Bal) {
      ERC20(pram.myTokenAddr).transfer(_to, Bal);
    } else {
      ERC20(pram.myTokenAddr).transfer(_to, _amount);
    }
}

    function getMultiplier(uint tokenId, uint256 _from, uint256 _to) public view returns (uint256) {
        TokenParam storage pram = tokenInfo[tokenId];
        uint start = startBlock[tokenId];
        uint bonusEndBlock = start.add(pram.blkNum1st);

        if(_to <= bonusEndBlock.add(pram.blkNum2nd)){
            if(_to <= bonusEndBlock){
                if(pram.blkNum1st == 0){
                    return 0;
                }
                else{
                    return
_to.sub(_from).mul(pram.amount1st).div(pram.blkNum1st).mul(100).div(pram.feeRate.add(100));
                }
            }
            else if(_from >= bonusEndBlock){
                if(pram.blkNum2nd == 0){
                    return 0;
                }
                else{
```

```
                                        return
_to.sub(_from).mul(pram.amount2nd).div(pram.blkNum2nd).mul(100).div(pram.feeRate.add(100));
                                }
                        }
                        else{
                                uint first;
                                uint sec;
                                if(pram.blkNum1st == 0){
                                        first = 0;
                                }
                                else{
                                        first                                   =
bonusEndBlock.sub(_from).mul(pram.amount1st).div(pram.blkNum1st);
                                }
                                if(pram.blkNum2nd == 0){
                                        sec = 0;
                                }
                                else{
                                        sec                                     =
_to.sub(bonusEndBlock).mul(pram.amount2nd).div(pram.blkNum2nd);
                                }
                           return first.add(sec).mul(100).div(pram.feeRate.add(100));
                        }
                }
                else{
                        if(pram.blkNum3rd == 0){
                                return 0;
                        }
                        uint blockHalfstart = bonusEndBlock.add(pram.blkNum2nd);
                        uint num = _to.sub(blockHalfstart).div(pram.blkNum3rd).add(1);
                        uint perBlock = pram.amount3rd.div(2 ** num).div(pram.blkNum3rd);
                        return _to.sub(_from).mul(perBlock).mul(100).div(pram.feeRate.add(100));
                }
        }

        function dev(uint tokenId, address _devAddr) public {//knownsec// renew dev address
            require(msg.sender == tokenInfo[tokenId].devAddr, "dev: wut?");
            tokenInfo[tokenId].devAddr = _devAddr;
        }


        function viewPoolInfo(uint tokenId) public view returns (PoolInfo[] memory){
                uint256 length = poolNum[tokenId];
                PoolInfo[] memory ret = new PoolInfo[](length);
                for (uint256 pid = 0; pid < length; ++pid) {
            ret[pid] = poolInfo[tokenId][pid];
```

```
    }
            return ret;
        }

        function viewTokenInfo() public view returns (TokenParam[] memory){
            TokenParam [] memory ret = new TokenParam[](tokenIndex);
            for(uint256 index = 0; index < tokenIndex; ++index){
                ret[index]=tokenInfo[index];
            }
            return ret;
        }
}
```

6. Appendix B:Security risks level standard

level standard of loophole rating in smart contract	
Level	**Descriptions**
High risk	Vulnerabilities that can directly damage token contracts or lead to assets loss, such as overflow that can invalidate the value of tokens, fake recharge that can cause token loss in exchanges, and re-entrancy which can cause ETH or token loss in contract account and so on. Vulnerabilities that can cause the ownership loss of token in contracts, such as defects in the access control of key functions, call injection leading to bypassing of access control of key functions and so on. Vulnerabilities that can result in the lack of smooth operation of token contract, such as the loophole of denial of service caused by ETH sent to a malicious address or gas exhaustion.
Medium risk	High-risk vulnerabilities that can only be triggered by specific address, such as overflow triggered by a token contract owner, access control defects of non-key functions, logical design defects that can cause indirect capital losses, etc.
Low risk	Vulnerabilities that are difficult to trigger, and vulnerabilities with limited damage after being triggered, such as numerical overflow vulnerabilities caused by a large number of ETH or tokens. Vulnerabilities that make attackers unable to directly profit after numerical overflow is triggered, and transaction sequential dependency triggered by high gas fee specified.

7. Appendix C: Introduction to smart contract security audit tools

7.1 Manticore

Manticore is a symbolic execution tool for analysis of smart contracts and binaries. It includes a symbolic Ethereum Virtual Machine (EVM), EVM disassembler/assembler, and an intuitive interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, used in the visual disassembler, Bit of Traits of Bits in EVM bytecode. Similar to binary files, Manticore provides a simple command-line interface and a Python API for EVM byte-code analysis.

7.2 Oyente

Oyente is an analysis tool for smart contract. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequential dependency, and so on. What is more convenient is the modular design of Oyente, which enables advanced users to insert their own detection logic to check customized attributes in their contracts.

7.3 securify.sh

Securify can verify common security issues in Ethereum smart contracts, such as transaction disorders and the lack of input verification. Fully automated, it analyzes all possible execution paths of the program. In addition, Securify is also equipped with specific language for vulnerability specification, which updates Securify on current issues such as security and reliability.

7.4 Echidna

Echidna is a Haskell program designed for fuzzing/property-based testing of Ethereum smarts contracts.

7.5 MAIAN

MAIAN is an automated tool used for vulnerability detection in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish transactions to find errors.

7.6 Ethersplay

Ethersplay is an EVM disassembler, which contains relevant analysis tools.

7.7 Ida-evm

Ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.8 Remix-ide

Remix is a browser-based compiler and IDE, and it enables users to create Ethereum contracts and debug transactions via Solidity language.

7.9 Tool kits of blockchain security for Knownsec auditors

The tool kits are developed, collected and used by Knownsec testing engineers. It contains automatic testing tools exclusive to testers, self-developed tools, scripts or utilization tools and so on.

KNOWNSEC

Beijing Knownsec Information Tech.CO.,LTD

Telephone +86(10)400 060 9587

Email sec@knownsec.com

Official website www.knownsec.com

Address 2509, block T2-B, WangJing SOHO, Chaoyang District, Beijing