KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: **KnownSec blockchain security research team**

Release notes

Revised content	Time	Revised by	Version number
Written document	20210120	KnownSec blockchain security research team	V1.0

Document information

Document name	Document version number	Document number	Confidentiality level
MDX Smart Contract Audit Report	V1.0	83d405ffd6f546feae3eafa052424d88	**Open project Team**

The statement

KnownSec only issues this report on the facts that have occurred or exist before the issuance of this report, and shall assume the corresponding responsibility therefor.KnownSec is not in a position to judge the security status of its smart contract and does not assume responsibility for the facts that occur or exist after issuance.The security audit analysis and other contents of this report are based solely on the documents and information provided by the information provider to KnownSec as of the issuance of this report.KnownSec assumes that the information provided was not missing, altered, truncated or suppressed.If the information provided is missing, altered, deleted, concealed or reflected in a way inconsistent with the actual situation, KnownSec shall not be liable for any loss or adverse effect caused thereby.

Directory

1. Review

The effective test time of this report is from January 15, 2021 to January 20, 2021. During this period, the security and standardization of the MDX smart contract code will be audited and used as the statistical basis for the report.

In this test, Known Chuangyu engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (see Chapter 3), and the comprehensive evaluation was passed.

The results of this smart contract security audit:Pass.

Since this testing process is carried out in a non-production environment, all codes are up-to-date backups, and the testing process is communicated with the relevant interface person, and relevant testing operations are carried out under the controllable operational risk to avoid production in the testing process Operational risk, code security risk.

Report information of this audit:

Report No:83d405ffd6f546feae3eafa052424d88

Report query address link:

https://attest.im/attestation/searchResult?qurey=83d405ffd6f546feae3eafa052424d88

Target information of this audit:

entry	description	
Token name	MDX	
Code type	Token code, HECO smart contract code, DEFI code, oracle code	
Code language	solidity	
Contract address	factory	0xb0b670fc1F7724119963018DB0BfA86aDb22d941
	router	0xED7d5F38C79115ca12fe6C0041abb22F0A06C300

	initcode	0x2ad889f82040abccb2649ea6a874796c1601fb67f91a747a80e08860c73ddf24
	MDXToken	0x25D2e80cB6B86881Fd7e07dd263Fb79f4AbE033c
	HecoPool	0xFB03e11D93632D97a8981158A632Dd5986F5E909
	swapMining	0x7373c42502874C88954bDd6D50b53061F018422e
	teamTimeLock	0xa3FD9758323C8A86292B55702F631c81283c9B79
	InvestorsTimeLock	0xa6FE654241140469d1757A5bB8Ee844325059569
	brandTimeLock	0x465D246233Ba20e7cfc95743B5d073BE8A7746B0

Contract document and hash:

Contract documents	MD5
GovernorAlpha.sol	07C29F446ADE2D2FFAA385EC13A12401
Timelock.sol	1820EB1AFE7CA05449FF4DD4BF21F437
Factory.sol	C4C1CC0E6CC3CDCC0A3CB6B5EBEEF8A2
HecoPool.sol	DDC0118D027CD6A4EBAFF70DBEA8A4DF
MdxTokenHeco.sol	309FC357E9438206EBF98E33EDF5FF3B
Router.sol	F75DC33F01CA7C4ED402DA8CEB72A56C
SwapMining.sol	44AC444A9C615C18714034274EBDBECC
IERC20.sol	3DCC72B8015697E5C65BA38C68AF9B2B
IMdexFactory.sol	4570CB06DF4D26AD2F84D35102E41EEC
IMdexPair.sol	B1F95827BE79AD5AA90F98C5A08A832E
IMdx.sol	5ADE6BE9BB228A2DA14DAA60ECA23B2B
SafeMath.sol	91BA36AD2E6D077B3A84EE30C78927A2

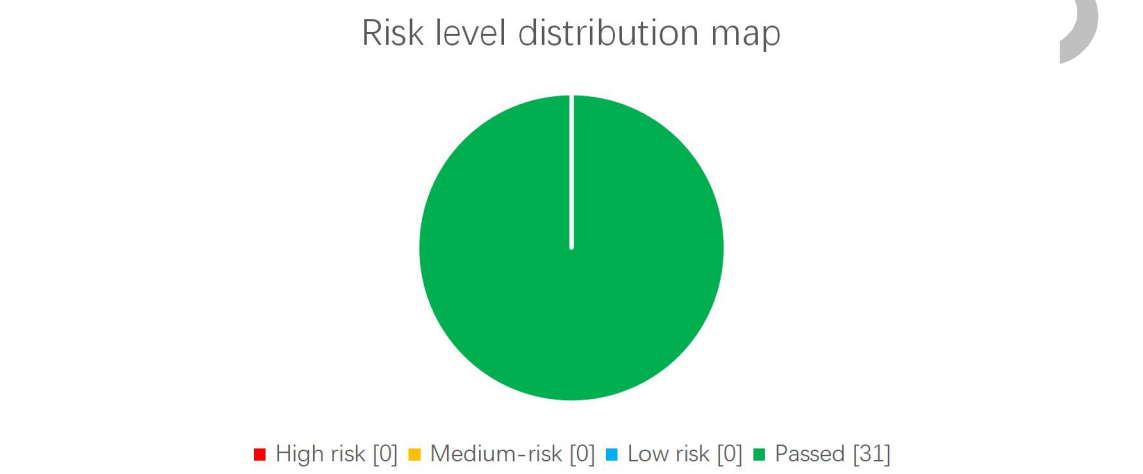
CoinChef.sol	A8D939346D0E747AA487BD3EBAF82F50
MdxToken.sol	6BAABBFF109099F7BB31C40134CCD7CF
Oracle.sol	5F082723070918B46DDCBEBE69CE6785

2. Code vulnerability analysis

2.1 Vulnerability level distribution

This vulnerability risk is calculated by level:

Statistics on the number of security risk levels			
High Risk	Medium Risk	Low Risk	Pass
0	0	0	31



2.2 Summary of audit results

Audit results			
Audit item	Audit content	status	description
Business security testing	Mortgage proof LP Token functions	pass	After testing, there are no safety issues.
	Pool contract pledge function	pass	After testing, there are no safety issues.
	Pool contract drawdown function	pass	After testing, there are no safety issues.
	Liquidity mining reward function	pass	After testing, there are no safety issues.
Basic code vulnerability detection	Compiler version security	pass	After testing, there are no safety issues.
	Redundant code	pass	After testing, there are no safety issues.
	Use of safe arithmetic library	pass	After testing, there are no safety issues.
	Not recommended encoding	pass	After testing, there are no safety issues.
	Reasonable use of require/assert	pass	After testing, there are no safety issues.
	fallback function safety	pass	After testing, there are no safety issues.
	tx.origin authentication	pass	After testing, there are no safety issues.
	owner permission control	pass	After testing, there are no safety issues.
	Gas consumption detection	pass	After testing, there are no safety issues.
	call injection attack	pass	After testing, there are no safety issues.
	Low-level function safety	pass	After testing, there are no safety issues.
	Vulnerabilities in issuing additional tokens	pass	After testing, there are no safety issues.
	Access control defect detection	pass	After testing, there are no safety issues.

	Numerical overflow detection	pass	After testing, there are no safety issues.
	Arithmetic accuracy error	pass	After testing, there are no safety issues.
	Wrong use of random number detection	pass	After testing, there are no safety issues.
	Unsafe interface use	pass	After testing, there are no safety issues.
	Variable coverage	pass	After testing, there are no safety issues.
	Uninitialized storage pointer	pass	After testing, there are no safety issues.
	Return value call verification	pass	After testing, there are no safety issues.
	Transaction order dependency detection	pass	After testing, there are no safety issues.
	Timestamp dependent attack	pass	After testing, there are no safety issues.
	Denial of service attack detection	pass	After testing, there are no safety issues.
	Fake recharge vulnerability detection	pass	After testing, there are no safety issues.
	Reentry attack detection	pass	After testing, there are no safety issues.
	Replay attack detection	pass	After testing, there are no safety issues.
	Rearrangement attack detection	pass	After testing, there are no safety issues.

3. Business Security Testing

3.1. Mortgage proof LP Token functions [Pass]

Audit analysis: The project contract uses the MdexERC20 contract as the LP Token in Factory.sol. According to the audit, the token contract has a reasonable design and correct authority control.

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

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

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

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

```
    function approve(address spender, uint value) external returns (bool) {// knownsec External authorized to the sender user
        _approve(msg.sender, spender, value);
        return true;
    }

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

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

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external {// knownsec External approval
        require(deadline >= block.timestamp, 'MdexSwap: EXPIRED');// knownsec Executable at the end
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline))
            )
```

```
            );
            address recoveredAddress = ecrecover(digest, v, r, s);
            require(recoveredAddress != address(0) && recoveredAddress == owner,
'MdexSwap: INVALID_SIGNATURE');
            _approve(owner, spender, value);
        }
```

Safety advice: None.

3.2. Pool contract pledge function[Pass]

Audit analysis: In the project contract, HecoPool is used as the token pool contract to manage token deposits and withdrawals, and deposit is used as the initial call method of pledge. After audit, the function is designed reasonably and the authority control is correct.

```
    function deposit(uint256 _pid, uint256 _amount) public notPause {// knownsec Stake LP to
HecoPool
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            depositMdxAndToken(_pid, _amount, msg.sender);
        } else {
            depositMdx(_pid, _amount, msg.sender);
        }
    }

    function depositMdxAndToken(uint256 _pid, uint256 _amount, address _user) private {//
knownsec Pledge Mdx and Sushi private
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);// knownsec Pool update
        if (user.amount > 0) {
```

```
            uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));//knownsec Starting value calculation
            IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], 0);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256 tokenPending = user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            if (pool.totalAmount == 0) {
                IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], _amount);
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            } else {
                uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
                IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], _amount);
                uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
                pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            }
        }
```

```
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }

    function depositMdx(uint256 _pid, uint256 _amount, address _user) private {// knownsec MDX Pledge private
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }
```

Safety advice: None.

3.3. Pool contract drawdown function[Pass]

Audit analysis: In the project contract, HecoPool is used as the token pool contract to manage token deposits and withdrawals, and withdraw is used as the initial

calling method of pledge. After audit, the function is designed reasonably and the authority control is correct.

```
    function withdraw(uint256 _pid, uint256 _amount) public notPause {// knownsec Unsuspended Executable Public LPToken in the retracement pool
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            withdrawMdxAndToken(_pid, _amount, msg.sender);
        } else {
            withdrawMdx(_pid, _amount, msg.sender);
        }
    }

    function withdrawMdxAndToken(uint256 _pid, uint256 _amount, address _user) private {// knownsec Private Mdx and Sushi in the retracement pool
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawMdxAndToken: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeMdxTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefHeco(multLpChef).withdraw(poolCorrespond[_pid], _amount);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256 tokenPending = user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
```

```
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    function withdrawMdx(uint256 _pid, uint256 _amount, address _user) private {// knownsec
Mdx in private drawdown pool
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawMdx: not good");
        updatePool(_pid);
        uint256                                pendingAmount                                =
user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeMdxTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }
```

Safety advice: None.

3.4. Liquidity mining reward function[Pass]

Audit analysis: The project contract uses SwapMining as the reward contract of the liquidity provider in SwapMining.sol, and uses takerWithdraw to collect the rewards obtained by the user. After audit, the token contract has a reasonable design and correct authority control.

```
function takerWithdraw() public {// knownsec All transaction rewards of this user in the pool are withdrawn publicly
        uint256 userSub;
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            PoolInfo storage pool = poolInfo[pid];
            UserInfo storage user = userInfo[pid][msg.sender];
            if (user.quantity > 0) {
                mint(pid);
                // The reward held by the user in this pool
                uint256 userReward = pool.allocMdxAmount.mul(user.quantity).div(pool.quantity);
                pool.quantity = pool.quantity.sub(user.quantity);
                pool.allocMdxAmount = pool.allocMdxAmount.sub(userReward);
                user.quantity = 0;
                user.blockNumber = block.number;
                userSub = userSub.add(userReward);
            }
        }
        if (userSub <= 0) {
            return;
        }
        mdx.transfer(msg.sender, userSub);
    }
```

Safety advice: None.

4. Basic code vulnerability detection

4.1. Compiler version security [Pass]

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

Test result: After testing, the smart contract code has a compiler version 0.5.16 or higher, and there is no such security issue.

Safety advice: None.

4.2. Redundant code [Pass]

Check whether the contract code implementation contains redundant code

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

Safety advice: None.

4.3. The use of safe arithmetic library [Pass]

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

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

Safety advice: None.

4.4. Unrecommended encoding method [Pass]

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

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

Safety advice: None.

4.5. Reasonable use of require/assert [Pass]

Check the rationality of the use of require and assert statements in the contract code implementation

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

Safety advice: None.

4.6. Fallback function safety [Pass]

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

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

Safety advice: None.

4.7. tx.origin authentication [Pass]

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks like phishing.

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

Safety advice: None.

4.8. Owner permission control [Pass]

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

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

Safety advice: None.

4.9. Gas consumption detection [Pass]

Check whether the consumption of gas exceeds the maximum block limit

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

Safety advice: None.

4.10. Call injection attack [Pass]

When calling the call function, strict permission control should be done, or the function called by the call should be written dead.

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

Safety advice: None.

4.11. Low-level function security [Pass]

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function

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

Safety advice: None.

4.12. Vulnerabilities in the issuance of additional tokens [Pass]

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

Test result: After testing, the mint method in the MdxTokenHeco.sol file in the smart contract code, Minter uses it to increase the total amount of tokens:

```
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {// knownsec Minter Available mint
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }
...
...
    function _mint(address account, uint256 amount) internal override virtual {
        super._mint(account, amount);

        // add delegates to the minter
        _moveDelegates(address(0), _delegates[account], amount);
    }
```

Security advice: This issue is not a security issue, but some exchanges will restrict the use of the additional issuance function. The specific situation depends on the requirements of the exchange and the specific project business.

4.13. Access control defect detection [Pass]

Different functions in the contract should set reasonable permissions

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

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

contract code.

Safety advice: None.

4.14. Numerical overflow detection [Pass]

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

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

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

Safety advice: None.

4.15. Arithmetic accuracy error [Pass]

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages—Solidity does not float. Point type, and all the numerical

calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same-level calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

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

Safety advice: None.

4.16. Incorrect use of random numbers [Pass]

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access obviously unpredictable values, such as block.number and block.timestamp, they are usually either more public than they appear or are affected by miners. These random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

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

Safety advice: None.

4.17. Unsafe interface usage [Pass]

Check whether unsafe interfaces are used in the contract code implementation

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

Safety advice: None.

4.18. Variable coverage [Pass]

Check whether there are security issues caused by variable coverage in the contract code implementation

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

Safety advice: None.

4.19. Uninitialized storage pointer [Pass]

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Test result: After testing, the smart contract code does not have this problem.

Safety advice: None.

4.20. Return value call verification [Pass]

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

There are transfer(), send(), call.value() and other currency transfer methods in Solidity, which can all be used to send HT to a certain address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be passed for calling (can be By passing in the gas_value parameter to limit), it cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to HT sending failure.

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

Safety advice: None.

4.21. Transaction order dependence [Pass]

Since miners always obtain gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since

the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

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

Safety advice: None.

4.22. Timestamp dependency attack [Pass]

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it is only necessary to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that rely on timestamps in the contract code implementation

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

Safety advice: None.

4.23. Denial of service attack [Pass]

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of a smart contract, including malicious behavior as a transaction receiver, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

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

Safety advice: None.

4.24. Fake recharge loopholes [Pass]

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, it enters the else logic part and returns false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in the scene of sensitive functions such as transfer.

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

Safety advice: None.

4.25. Reentry attack detection [Pass]

Re-entry vulnerabilities are the most famous Ethereum smart contract vulnerabilities.

The call.value() function in Solidity consumes all the gas it receives when it is used to send HT. When the call.value() function is called to send HT before the actual reduction of the balance of the sender's account, There is a risk of reentry attacks.

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

Safety advice: None.

4.26. Replay attack detection [Pass]

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario 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.

Safety advice: None.

4.27. Rearrangement attack detection [Pass]

A rearrangement attack is when miners or other parties try to "compete" with

smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Test result: After testing, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

5. Appendix A: Contract code

Source code for this test:

GovernorAlpha.sol

```
pragma solidity ^0.6.12;
pragma experimental ABIEncoderV2;

import "../heco/MdxTokenHeco.sol";

contract GovernorAlpha {/// knownsec    Governance contract
    /// @notice The name of this contract
    string public constant name = "MDX Governor Alpha";

    /// @notice The number of votes in support of a proposal required in order for a quorum to be reached and for a vote to succeed
    function quorumVotes() public view returns (uint) {return mdx.totalSupply() / 25;} // 400,000 = 4% of MDX

    /// @notice The number of votes required in order for a voter to become a proposer
    function proposalThreshold() public view returns (uint) {return mdx.totalSupply() / 100;} // 100,000 = 1% of MDX

    /// @notice The maximum number of actions that can be included in a proposal
    function proposalMaxOperations() public pure returns (uint) {return 10;} // 10 actions

    /// @notice The delay before voting on a proposal may take place, once proposed
    function votingDelay() public pure returns (uint) {return 1;} // 1 block

    /// @notice The duration of voting on a proposal, in blocks
    function votingPeriod() public pure returns (uint) {return 86400;} // ~3 days in blocks (assuming 3s blocks)

    /// @notice The address of the MDX Protocol Timelock
    TimelockInterface public timelock;

    /// @notice The address of the MDX governance token
    MdxToken public mdx;

    /// @notice The address of the Governor Guardian
    address public guardian;

    /// @notice The total number of proposals
    uint public proposalCount;

    struct Proposal {
        /// @notice Unique id for looking up a proposal
        uint id;

        /// @notice Creator of the proposal
        address proposer;

        /// @notice The timestamp that the proposal will be available for execution, set once the vote succeeds
        uint eta;

        /// @notice the ordered list of target addresses for calls to be made
        address[] targets;

        /// @notice The ordered list of values (i.e. msg.value) to be passed to the calls to be made
        uint[] values;

        /// @notice The ordered list of function signatures to be called
        string[] signatures;

        /// @notice The ordered list of calldata to be passed to each call
        bytes[] calldatas;

        /// @notice The block at which voting begins: holders must delegate their votes prior to this block
        uint startBlock;

        /// @notice The block at which voting ends: votes must be cast prior to this block
        uint endBlock;

        /// @notice Current number of votes in favor of this proposal
        uint forVotes;

        /// @notice Current number of votes in opposition to this proposal
        uint againstVotes;

        /// @notice Flag marking whether the proposal has been canceled
        bool canceled;

        /// @notice Flag marking whether the proposal has been executed
        bool executed;
```

```
        /// @notice Receipts of ballots for the entire set of voters
        mapping(address => Receipt) receipts;
    }

    /// @notice Ballot receipt record for a voter
    struct Receipt {
        /// @notice Whether or not a vote has been cast
        bool hasVoted;

        /// @notice Whether or not the voter supports the proposal
        bool support;

        /// @notice The number of votes the voter had, which were cast
        uint256 votes;
    }

    /// @notice Possible states that a proposal may be in
    enum ProposalState {
        Pending,
        Active,
        Canceled,
        Defeated,
        Succeeded,
        Queued,
        Expired,
        Executed
    }

    /// @notice The official record of all proposals ever proposed
    mapping(uint => Proposal) public proposals;

    /// @notice The latest proposal for each proposer
    mapping(address => uint) public latestProposalIds;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    /// @notice The EIP-712 typehash for the ballot struct used by the contract
    bytes32 public constant BALLOT_TYPEHASH = keccak256("Ballot(uint256 proposalId,bool support)");

    /// @notice An event emitted when a new proposal is created
    event ProposalCreated(uint id, address proposer, address[] targets, uint[] values, string[] signatures, bytes[] calldatas, uint startBlock, uint endBlock, string description);

    /// @notice An event emitted when a vote has been cast on a proposal
    event VoteCast(address voter, uint proposalId, bool support, uint votes);

    /// @notice An event emitted when a proposal has been canceled
    event ProposalCanceled(uint id);

    /// @notice An event emitted when a proposal has been queued in the Timelock
    event ProposalQueued(uint id, uint eta);

    /// @notice An event emitted when a proposal has been executed in the Timelock
    event ProposalExecuted(uint id);

    constructor(address timelock_, address mdx_, address guardian_) public {
        timelock = TimelockInterface(timelock_);
        mdx = MdxToken(mdx_);
        guardian = guardian_;
    }

    function propose(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas, string memory description) public returns (uint) {
        require(mdx.getPriorVotes(msg.sender, sub256(block.number, 1)) > proposalThreshold(), "GovernorAlpha::propose: proposer votes below proposal threshold");
        require(targets.length == values.length && targets.length == signatures.length && targets.length == calldatas.length, "GovernorAlpha::propose: proposal function information arity mismatch");
        require(targets.length != 0, "GovernorAlpha::propose: must provide actions");
        require(targets.length <= proposalMaxOperations(), "GovernorAlpha::propose: too many actions");

        uint latestProposalId = latestProposalIds[msg.sender];
        if (latestProposalId != 0) {
            ProposalState proposersLatestProposalState = state(latestProposalId);
            require(proposersLatestProposalState != ProposalState.Active, "GovernorAlpha::propose: one live proposal per proposer, found an already active proposal");
            require(proposersLatestProposalState != ProposalState.Pending, "GovernorAlpha::propose: one live proposal per proposer, found an already pending proposal");
        }

        uint startBlock = add256(block.number, votingDelay());
        uint endBlock = add256(startBlock, votingPeriod());

        proposalCount++;
        Proposal memory newProposal = Proposal({
```

```
            id : proposalCount,
            proposer : msg.sender,
            eta : 0,
            targets : targets,
            values : values,
            signatures : signatures,
            calldatas : calldatas,
            startBlock : startBlock,
            endBlock : endBlock,
            forVotes : 0,
            againstVotes : 0,
            canceled : false,
            executed : false
            });

            proposals[newProposal.id] = newProposal;
            latestProposalIds[newProposal.proposer] = newProposal.id;

            emit ProposalCreated(newProposal.id, msg.sender, targets, values, signatures, calldatas, startBlock, endBlock, description);
            return newProposal.id;
    }

    function queue(uint proposalId) public {
            require(state(proposalId) == ProposalState.Succeeded, "GovernorAlpha::queue: proposal can only be queued if it is succeeded");
            Proposal storage proposal = proposals[proposalId];
            uint eta = add256(block.timestamp, timelock.delay());
            for (uint i = 0; i < proposal.targets.length; i++) {
                _queueOrRevert(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], eta);
            }
            proposal.eta = eta;
            emit ProposalQueued(proposalId, eta);
    }

    function _queueOrRevert(address target, uint value, string memory signature, bytes memory data, uint eta) internal {
            require(!timelock.queuedTransactions(keccak256(abi.encode(target, value, signature, data, eta))), "GovernorAlpha::_queueOrRevert: proposal action already queued at eta");
            timelock.queueTransaction(target, value, signature, data, eta);
    }

    function execute(uint proposalId) public payable {
            require(state(proposalId) == ProposalState.Queued, "GovernorAlpha::execute: proposal can only be executed if it is queued");
            Proposal storage proposal = proposals[proposalId];
            proposal.executed = true;
            for (uint i = 0; i < proposal.targets.length; i++) {
                timelock.executeTransaction{value : proposal.values[i]}(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], proposal.eta);
            }
            emit ProposalExecuted(proposalId);
    }

    function cancel(uint proposalId) public {
            ProposalState state = state(proposalId);
            require(state != ProposalState.Executed, "GovernorAlpha::cancel: cannot cancel executed proposal");

            Proposal storage proposal = proposals[proposalId];
            require(msg.sender == guardian || mdx.getPriorVotes(proposal.proposer, sub256(block.number, 1)) < proposalThreshold(), "GovernorAlpha::cancel: proposer above threshold");

            proposal.canceled = true;
            for (uint i = 0; i < proposal.targets.length; i++) {
                timelock.cancelTransaction(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], proposal.eta);
            }

            emit ProposalCanceled(proposalId);
    }

    function getActions(uint proposalId) public view returns (address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas) {
            Proposal storage p = proposals[proposalId];
            return (p.targets, p.values, p.signatures, p.calldatas);
    }

    function getReceipt(uint proposalId, address voter) public view returns (Receipt memory) {
            return proposals[proposalId].receipts[voter];
    }

    function state(uint proposalId) public view returns (ProposalState) {
            require(proposalCount >= proposalId && proposalId > 0, "GovernorAlpha::state: invalid proposal id");
            Proposal storage proposal = proposals[proposalId];
            if (proposal.canceled) {
```

```
            return ProposalState.Canceled;
        } else if (block.number <= proposal.startBlock) {
            return ProposalState.Pending;
        } else if (block.number <= proposal.endBlock) {
            return ProposalState.Active;
        } else if (proposal.forVotes <= proposal.againstVotes || proposal.forVotes < quorumVotes()) {
            return ProposalState.Defeated;
        } else if (proposal.eta == 0) {
            return ProposalState.Succeeded;
        } else if (proposal.executed) {
            return ProposalState.Executed;
        } else if (block.timestamp >= add256(proposal.eta, timelock.GRACE_PERIOD())) {
            return ProposalState.Expired;
        } else {
            return ProposalState.Queued;
        }
    }

    function castVote(uint proposalId, bool support) public {
        return _castVote(msg.sender, proposalId, support);
    }

    function castVoteBySig(uint proposalId, bool support, uint8 v, bytes32 r, bytes32 s) public {
        bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name)),
getChainId(), address(this)));
        bytes32 structHash = keccak256(abi.encode(BALLOT_TYPEHASH, proposalId, support));
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "GovernorAlpha::castVoteBySig: invalid signature");
        return _castVote(signatory, proposalId, support);
    }

    function _castVote(address voter, uint proposalId, bool support) internal {
        require(state(proposalId) == ProposalState.Active, "GovernorAlpha::_castVote: voting is closed");
        Proposal storage proposal = proposals[proposalId];
        Receipt storage receipt = proposal.receipts[voter];
        require(receipt.hasVoted == false, "GovernorAlpha::_castVote: voter already voted");
        uint256 votes = mdx.getPriorVotes(voter, proposal.startBlock);

        if (support) {
            proposal.forVotes = add256(proposal.forVotes, votes);
        } else {
            proposal.againstVotes = add256(proposal.againstVotes, votes);
        }

        receipt.hasVoted = true;
        receipt.support = support;
        receipt.votes = votes;

        emit VoteCast(voter, proposalId, support, votes);
    }

    function __acceptAdmin() public {
        require(msg.sender == guardian, "GovernorAlpha::__acceptAdmin: sender must be gov guardian");
        timelock.acceptAdmin();
    }

    function __abdicate() public {
        require(msg.sender == guardian, "GovernorAlpha::__abdicate: sender must be gov guardian");
        guardian = address(0);
    }

    function __queueSetTimelockPendingAdmin(address newPendingAdmin, uint eta) public {
        require(msg.sender == guardian, "GovernorAlpha::__queueSetTimelockPendingAdmin: sender must be
gov guardian");
        timelock.queueTransaction(address(timelock), 0, "setPendingAdmin(address)",
abi.encode(newPendingAdmin), eta);
    }

    function __executeSetTimelockPendingAdmin(address newPendingAdmin, uint eta) public {
        require(msg.sender == guardian, "GovernorAlpha::__executeSetTimelockPendingAdmin: sender must
be gov guardian");
        timelock.executeTransaction(address(timelock), 0, "setPendingAdmin(address)",
abi.encode(newPendingAdmin), eta);
    }

    function add256(uint256 a, uint256 b) internal pure returns (uint) {
        uint c = a + b;
        require(c >= a, "addition overflow");
        return c;
    }

    function sub256(uint256 a, uint256 b) internal pure returns (uint) {
        require(b <= a, "subtraction underflow");
        return a - b;
    }
```

```
        function getChainId() internal pure returns (uint) {
            uint chainId;
            assembly {chainId := chainid()}
            return chainId;
        }
}

interface TimelockInterface {
        function delay() external view returns (uint);

        function GRACE_PERIOD() external view returns (uint);

        function acceptAdmin() external;

        function queuedTransactions(bytes32 hash) external view returns (bool);

        function queueTransaction(address target, uint value, string calldata signature, bytes calldata data, uint eta) external returns (bytes32);

        function cancelTransaction(address target, uint value, string calldata signature, bytes calldata data, uint eta) external;

        function executeTransaction(address target, uint value, string calldata signature, bytes calldata data, uint eta) external payable returns (bytes memory);
}
```

Timelock.sol

```
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";

contract Timelock {// knownsec Proposal time lock contract
        using SafeMath for uint;

        event NewAdmin(address indexed newAdmin);
        event NewPendingAdmin(address indexed newPendingAdmin);
        event NewDelay(uint indexed newDelay);
        event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
        event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
        event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

        uint public constant GRACE_PERIOD = 14 days;
        uint public constant MINIMUM_DELAY = 2 days;
        uint public constant MAXIMUM_DELAY = 30 days;

        address public admin;
        address public pendingAdmin;
        uint public delay;

        mapping(bytes32 => bool) public queuedTransactions;


        constructor(address admin_, uint delay_) public {
            require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum delay.");
            require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum delay.");

            admin = admin_;
            delay = delay_;
        }

        receive() external payable {}

        function setDelay(uint delay_) public {
            require(msg.sender == address(this), "Timelock::setDelay: Call must come from Timelock.");
            require(delay_ >= MINIMUM_DELAY, "Timelock::setDelay: Delay must exceed minimum delay.");
            require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum delay.");
            delay = delay_;

            emit NewDelay(delay);
        }

        function acceptAdmin() public {
            require(msg.sender == pendingAdmin, "Timelock::acceptAdmin: Call must come from pendingAdmin.");
            admin = msg.sender;
            pendingAdmin = address(0);

            emit NewAdmin(admin);
        }

        function setPendingAdmin(address pendingAdmin_) public {// knownsec The proposal is sent through the admin transfer itself
            require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from Timelock.");
```

```
        pendingAdmin = pendingAdmin_;

        emit NewPendingAdmin(pendingAdmin);
    }

    function queueTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public returns (bytes32) {
        require(msg.sender == admin, "Timelock::queueTransaction: Call must come from admin.");
        require(eta >= getBlockTimestamp().add(delay), "Timelock::queueTransaction: Estimated execution block must satisfy delay.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = true;

        emit QueueTransaction(txHash, target, value, signature, data, eta);
        return txHash;
    }

    function cancelTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public {
        require(msg.sender == admin, "Timelock::cancelTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = false;

        emit CancelTransaction(txHash, target, value, signature, data, eta);
    }

    function executeTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public payable returns (bytes memory) {
        require(msg.sender == admin, "Timelock::executeTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        require(queuedTransactions[txHash], "Timelock::executeTransaction: Transaction hasn't been queued.");
        require(getBlockTimestamp() >= eta, "Timelock::executeTransaction: Transaction hasn't surpassed time lock.");
        require(getBlockTimestamp() <= eta.add(GRACE_PERIOD), "Timelock::executeTransaction: Transaction is stale.");

        queuedTransactions[txHash] = false;

        bytes memory callData;

        if (bytes(signature).length == 0) {
            callData = data;
        } else {
            callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
        }

        // solium-disable-next-line security/no-call-value
        (bool success, bytes memory returnData) = target.call{value : value}(callData);
        require(success, "Timelock::executeTransaction: Transaction execution reverted.");

        emit ExecuteTransaction(txHash, target, value, signature, data, eta);

        return returnData;
    }

    function getBlockTimestamp() internal view returns (uint) {
        // solium-disable-next-line security/no-block-members
        return block.timestamp;
    }
}
```

Factory.sol

```
pragma solidity >=0.5.0 <0.8.0;

import "../library/SafeMath.sol";
import "../interface/IERC20.sol";
import "../interface/IMdexFactory.sol";
import "../interface/IMdexPair.sol";

interface IHswapV2Callee {
    function hswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) external;
}

interface IMdexERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);
```

```
    function totalSupply() external view returns (uint);

    function balanceOf(address owner) external view returns (uint);

    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);

    function transfer(address to, uint value) external returns (bool);

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

    function DOMAIN_SEPARATOR() external view returns (bytes32);

    function PERMIT_TYPEHASH() external pure returns (bytes32);

    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
}

contract MdexERC20 is IMdexERC20 {// knownsec MdexERC20 contract
    using SafeMath for uint;

    string public constant name = 'HSwap LP Token';
    string public constant symbol = 'HMDX';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
    mapping(address => uint) public nonces;

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

    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)'),
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }

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

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

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

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

    function approve(address spender, uint value) external returns (bool) {// knownsec External authorized to the sender user
        _approve(msg.sender, spender, value);
        return true;
    }
```

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

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

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external {// knownsec External approval
        require(deadline >= block.timestamp, 'MdexSwap: EXPIRED');// knownsec Executable at the end
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'MdexSwap: INVALID_SIGNATURE');
        _approve(owner, spender, value);
    }
}

contract MdexPair is IMdexPair, MdexERC20 {// knownsec MdexPair contract
    using SafeMath  for uint;
    using UQ112x112 for uint224;

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

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

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

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

    uint private unlocked = 1;
    modifier lock() {// knownsec Prevent re-entry lock
        require(unlocked == 1, 'MdexSwap: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }

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

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

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

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

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

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

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

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

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply;// knownsec Total supply
        // gas savings, must be defined here since totalSupply can update in _mintFee
        if (_totalSupply == 0) {
            liquidity = SafeMath.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);// knownsec
Liquidity calculation
            mint(address(0), MINIMUM_LIQUIDITY);// knownsec Lock the first lowest liquidity token
            // permanently lock the first MINIMUM_LIQUIDITY tokens
        } else {
            liquidity = SafeMath.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) /
_reserve1);
        }
        require(liquidity > 0, 'MdexSwap: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

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

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

```
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];

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

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

    // this low-level function should be called from a contract which performs important safety checks
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'MdexSwap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'MdexSwap:
INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        {// scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'MdexSwap: INVALID_TO');
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out);
            // optimistically transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out);
            // optimistically transfer tokens
            if (data.length > 0) IHswapV2Callee(to).hswapV2Call(msg.sender, amount0Out, amount1Out,
data);
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'MdexSwap: INSUFFICIENT_INPUT_AMOUNT');
        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
            uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000 **
2), 'MdexSwap: K');
        }

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

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

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)), reserve0,
reserve1);
    }

    function price(address token, uint256 baseDecimal) public view returns (uint256) {// knownsec // 以
baseDecimal Accuracy calculation of token price requires token0 or token1
        if ((token0 != token && token1 != token) || 0 == reserve0 || 0 == reserve1) {
            return 0;
        }
        if (token0 == token) {
            return uint256(reserve1).mul(baseDecimal).div(uint256(reserve0));
        } else {
            return uint256(reserve0).mul(baseDecimal).div(uint256(reserve1));
        }
    }
```

```
}

contract MdexFactory is IMdexFactory {// knownsec MdexFactory contract
    using SafeMath for uint256;
    address public feeTo;
    address public feeToSetter;
    uint256 public feeToRate;
    bytes32 public initCodeHash;

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

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

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
        initCodeHash = keccak256(abi.encodePacked(type(MdexPair).creationCode));
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external returns (address pair) {// knownsec Create a
matching contract
        require(tokenA != tokenB, 'MdexSwapFactory: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'MdexSwapFactory: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'MdexSwapFactory: PAIR_EXISTS');
        // single check is sufficient
        bytes memory bytecode = type(MdexPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IMdexPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair;
        // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

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

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'MdexSwapFactory: FORBIDDEN');
        require(_feeToSetter != address(0), "MdexSwapFactory: FeeToSetter is zero address");
        feeToSetter = _feeToSetter;
    }

    function setFeeToRate(uint256 _rate) external {
        require(msg.sender == feeToSetter, 'MdexSwapFactory: FORBIDDEN');
        require(_rate > 0, "MdexSwapFactory: FEE_TO_RATE_OVERFLOW");
        feeToRate = _rate.sub(1);
    }

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

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

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
    function quote(uint amountA, uint reserveA, uint reserveB) public pure returns (uint amountB) {
```

```
        require(amountA > 0, 'MdexSwapFactory: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'MdexSwapFactory: INSUFFICIENT_LIQUIDITY');
        amountB = amountA.mul(reserveB) / reserveA;
    }

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

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

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

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

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

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

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

HecoPool.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "../interface/IMdx.sol";

interface IMasterChefHeco {
    function pending(uint256 pid, address user) external view returns (uint256);

    function deposit(uint256 pid, uint256 amount) external;

    function withdraw(uint256 pid, uint256 amount) external;

    function emergencyWithdraw(uint256 pid) external;
}
```

```
contract HecoPool is Ownable {// knownsec Heco Pool contract
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _multLP;

    // Info of each user.
    struct UserInfo {// knownsec User information structure
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt.
        uint256 multLpRewardDebt; //multLp Reward debt.
    }

    // Info of each pool.
    struct PoolInfo {// knownsec Pool information structure
        IERC20 lpToken;                 // Address of LP token contract.
        uint256 allocPoint;             // How many allocation points assigned to this pool. MDXs to distribute per
block.
        uint256 lastRewardBlock;    // Last block number that MDXs distribution occurs.
        uint256 accMdxPerShare; // Accumulated MDXs per share, times 1e12.
        uint256 accMultLpPerShare; //Accumulated multLp per share
        uint256 totalAmount;        // Total amount of current pool deposit.
    }

    // The MDX Token!
    IMdx public mdx;
    // MDX tokens created per block.
    uint256 public mdxPerBlock;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
    // Corresponding to the pid of the multLP pool
    mapping(uint256 => uint256) public poolCorrespond;
    // pid corresponding address
    mapping(address => uint256) public LpOfPid;
    // Control mining
    bool public paused = false;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when MDX mining starts.
    uint256 public startBlock;
    // multLP MasterChef
    address public multLpChef;
    // multLP Token
    address public multLpToken;
    // How many blocks are halved
    uint256 public halvingPeriod = 14400;

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

    constructor(
        IMdx _mdx,
        uint256 _mdxPerBlock,
        uint256 _startBlock
    ) public {// knownsec    Initialize incoming _mdx _mdxPerBlock _startBlock
        mdx = _mdx;
        mdxPerBlock = _mdxPerBlock;
        startBlock = _startBlock;
    }

    function setHalvingPeriod(uint256 _block) public onlyOwner {// knownsec Owner Use to set halvingPeriod
        halvingPeriod = _block;
    }

    function poolLength() public view returns (uint256) {
        return poolInfo.length;
    }

    function addMultLP(address _addLP) public onlyOwner returns (bool) {// knownsec Owner Use addMultLP
        require(_addLP != address(0), "LP is the zero address");
        IERC20(_addLP).approve(multLpChef, uint256(- 1));
        return EnumerableSet.add(_multLP, _addLP);
    }

    function isMultLP(address _LP) public view returns (bool) {// knownsec Check if it is MultLP
        return EnumerableSet.contains(_multLP, _LP);
    }

    function getMultLPLength() public view returns (uint256) {// knownsec    Mult length acquisition
        return EnumerableSet.length(_multLP);
    }

    function getMultLPAddress(uint256 _pid) public view returns (address){
```

```
        require(_pid <= getMultLPLength() - 1, "not find this multLP");
        return EnumerableSet.at(_multLP, _pid);
    }

    function setPause() public onlyOwner {// knownsec Owner Use pause settings
        paused = !paused;
    }

    function setMultLP(address _multLpToken, address _multLpChef) public onlyOwner {// knownsec Owner Use
Set up MultLP
        require(_multLpToken != address(0) && _multLpChef != address(0), "is the zero address");
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
    }

    function replaceMultLP(address _multLpToken, address _multLpChef) public onlyOwner {// knownsec
OwnerUse to replace MultLP
        require(_multLpToken != address(0) && _multLpChef != address(0), "is the zero address");
        require(paused, "No mining suspension");
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
        uint256 length = getMultLPLength();
        while (length > 0) {
            address dAddress = EnumerableSet.at(_multLP, 0);
            uint256 pid = LpOfPid[dAddress];
            IMasterChefHeco(multLpChef).emergencyWithdraw(poolCorrespond[pid]);
            EnumerableSet.remove(_multLP, dAddress);
            length--;
        }
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {// knownsec
OwnerUse to add a new lp to the pool
        require(address(_lpToken) != address(0), "_lpToken is the zero address");
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
        lpToken : _lpToken,
        allocPoint : _allocPoint,
        lastRewardBlock : lastRewardBlock,
        accMdxPerShare : 0,
        accMultLpPerShare : 0,
        totalAmount : 0
        }));
        LpOfPid[address(_lpToken)] = poolLength() - 1;
    }

    // Update the given pool's MDX allocation point. Can only be called by the owner.
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {// knownsec Owner
Use to set the allocation point of the pool
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // The current pool corresponds to the pid of the multLP pool
    function setPoolCorr(uint256 _pid, uint256 _sid) public onlyOwner {// knownsec Owner use
        require(_pid <= poolLength() - 1, "not find this pool");
        poolCorrespond[_pid] = _sid;
    }

    function phase(uint256 blockNumber) public view returns (uint256) {// knownsec Calculating at the stage
Public view
        if (halvingPeriod == 0) {
            return 0;
        }
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function reward(uint256 blockNumber) public view returns (uint256) {// knownsec Calculating the reward
stage in which it is in public view
        uint256 _phase = phase(blockNumber);
        return mdxPerBlock.div(2 ** _phase);
    }

    function getMdxBlockReward(uint256 _lastRewardBlock) public view returns (uint256) {// knownsec
MdxBlock reward stage calculation Public view
```

```
        uint256 blockReward = 0;
        uint256 n = phase(_lastRewardBlock);
        uint256 m = phase(block.number);
        while (n < m) {
            n++;
            uint256 r = n.mul(halvingPeriod).add(startBlock);
            blockReward = blockReward.add((r.sub(_lastRewardBlock)).mul(reward(r)));
            _lastRewardBlock = r;
        }
        blockReward = blockReward.add((block.number.sub(_lastRewardBlock)).mul(reward(block.number)));
        return blockReward;
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools() public {// knownsec Update the reward variables of all pools
        uint256 length = poolInfo.length;// knownsec Pool number calculation
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {// knownsec Designated pool update public
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply;
        if (isMultLP(address(pool.lpToken))) {
            if (pool.totalAmount == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
            lpSupply = pool.totalAmount;
        } else {
            lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
        }
        uint256 blockReward = getMdxBlockReward(pool.lastRewardBlock);
        if (blockReward <= 0) {
            return;
        }
        uint256 mdxReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        bool minRet = mdx.mint(address(this), mdxReward);
        if (minRet) {
            pool.accMdxPerShare = pool.accMdxPerShare.add(mdxReward.mul(1e12).div(lpSupply));
        }
        pool.lastRewardBlock = block.number;
    }

    // View function to see pending MDXs on frontend.
    function pending(uint256 _pid, address r) external view returns (uint256, uint256){// knownsec View the MDX
external calls that have not been processed before
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            (uint256 mdxAmount, uint256 tokenAmount) = pendingMdxAndToken(_pid, _user);
            return (mdxAmount, tokenAmount);
        } else {
            uint256 mdxAmount = pendingMdx(_pid, _user);
            return (mdxAmount, 0);
        }
    }

    function pendingMdxAndToken(uint256 _pid, address _user) private view returns (uint256, uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accMdxPerShare = pool.accMdxPerShare;
        uint256 accMultLpPerShare = pool.accMultLpPerShare;
        if (user.amount > 0) {
            uint256 TokenPending = IMasterChefHeco(multLpChef).pending(poolCorrespond[_pid],
address(this));
            accMultLpPerShare = accMultLpPerShare.add(TokenPending.mul(1e12).div(pool.totalAmount));
            uint256 userPending =
user.amount.mul(accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getMdxBlockReward(pool.lastRewardBlock);
                uint256 mdxReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accMdxPerShare = accMdxPerShare.add(mdxReward.mul(1e12).div(pool.totalAmount));
                return (user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
            if (block.number == pool.lastRewardBlock) {
                return (user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
        }
```

```
        return (0, 0);
    }

    function pendingMdx(uint256 _pid, address _user) private view returns (uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accMdxPerShare = pool.accMdxPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (user.amount > 0) {
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getMdxBlockReward(pool.lastRewardBlock);
                uint256 mdxReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accMdxPerShare = accMdxPerShare.add(mdxReward.mul(1e12).div(lpSupply));
                return user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt);
            }
            if (block.number == pool.lastRewardBlock) {
                return user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt);
            }
        }
        return 0;
    }

    // Deposit LP tokens to HecoPool for MDX allocation.
    function deposit(uint256 _pid, uint256 _amount) public notPause {// knownsec Stake LP to HecoPool
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            depositMdxAndToken(_pid, _amount, msg.sender);
        } else {
            depositMdx(_pid, _amount, msg.sender);
        }
    }

    function depositMdxAndToken(uint256 _pid, uint256 _amount, address _user) private {// knownsec Pledge
Mdx and Sushi private
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);// knownsec Pool update
        if (user.amount > 0) {
            uint256                                    pendingAmount                                    =
user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));// knownsecStarting value
calculation
            IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], 0);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare                                                                      =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256                                    tokenPending                                     =
user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            if (pool.totalAmount == 0) {
                IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], _amount);
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            } else {
                uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
                IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], _amount);
                uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
                pool.accMultLpPerShare                                                                  =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            }
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }

    function depositMdx(uint256 _pid, uint256 _amount, address _user) private {// knownsec MDX Pledge
private
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256                                    pendingAmount                                    =
user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
```

```
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }

    // Withdraw LP tokens from HecoPool.
    function withdraw(uint256 _pid, uint256 _amount) public notPause {// knownsec Unsuspended Executable Public LPToken in the retracement pool
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            withdrawMdxAndToken(_pid, _amount, msg.sender);
        } else {
            withdrawMdx(_pid, _amount, msg.sender);
        }
    }

    function withdrawMdxAndToken(uint256 _pid, uint256 _amount, address _user) private {// knownsec Private Mdx and Sushi in the retracement pool
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawMdxAndToken: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeMdxTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefHeco(multLpChef).withdraw(poolCorrespond[_pid], _amount);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256 tokenPending =
user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    function withdrawMdx(uint256 _pid, uint256 _amount, address _user) private {// knownsec Mdx in private drawdown pool
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawMdx: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeMdxTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public notPause {// knownsec No suspension of execution, public emergency, no profit drawdown
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            emergencyWithdrawMdxAndToken(_pid, msg.sender);
        } else {
            emergencyWithdrawMdx(_pid, msg.sender);
        }
    }

    function emergencyWithdrawMdxAndToken(uint256 _pid, address _user) private {// knownsec Private Emergency Mdx and LPToken are not calculated
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 amount = user.amount;
        uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
```

```
            IMasterChefHeco(multLpChef).withdraw(poolCorrespond[_pid], amount);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            user.amount = 0;
            user.rewardDebt = 0;
            pool.lpToken.safeTransfer(_user, amount);
            pool.totalAmount = pool.totalAmount.sub(amount);
            emit EmergencyWithdraw(_user, _pid, amount);
        }

        function emergencyWithdrawMdx(uint256 _pid, address _user) private {// knownsec Private Emergency
Retracement of MDX
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_user];
            uint256 amount = user.amount;
            user.amount = 0;
            user.rewardDebt = 0;
            pool.lpToken.safeTransfer(_user, amount);
            pool.totalAmount = pool.totalAmount.sub(amount);
            emit EmergencyWithdraw(_user, _pid, amount);
        }

        // Safe MDX transfer function, just in case if rounding error causes pool to not have enough MDXs.
        function safeMdxTransfer(address _to, uint256 _amount) internal {// knownsec MDX transfer function, check
balance internal
            uint256 mdxBal = mdx.balanceOf(address(this));
            if (_amount > mdxBal) {
                mdx.transfer(_to, mdxBal);
            } else {
                mdx.transfer(_to, _amount);
            }
        }

        modifier notPause() {// 未暂停修饰器
            require(!paused, "Mining has been suspended");
            _;
        }
}
```

MdxTokenHeco.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";

abstract contract DelegateERC20 is ERC20 {
    /// @notice A record of each accounts delegate
    mapping (address => address) internal _delegates;

    /// @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    /// @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    /// @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256
chainId,address verifyingContract)");

    /// @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256
nonce,uint256 expiry)");

    /// @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;


    // support delegates mint
    function _mint(address account, uint256 amount) internal override virtual {
        super._mint(account, amount);

        // add delegates to the minter
        _moveDelegates(address(0), _delegates[account], amount);
    }
```

```
function _transfer(address sender, address recipient, uint256 amount) internal override virtual {
        super._transfer(sender, recipient, amount);
        _moveDelegates(_delegates[sender], _delegates[recipient], amount);
}


/**
 * @notice Delegate votes from `msg.sender` to `delegatee`
 * @param delegatee The address to delegate votes to
 */
function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
}

/**
  * @notice Delegates votes from signatory to `delegatee`
  * @param delegatee The address to delegate votes to
  * @param nonce The contract state required to match the signature
  * @param expiry The time at which to expire the signature
  * @param v The recovery byte of the signature
  * @param r Half of the ECDSA signature pair
  * @param s Half of the ECDSA signature pair
  */
function delegateBySig(
        address delegatee,
        uint nonce,
        uint expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
)
external
{
        bytes32 domainSeparator = keccak256(
                abi.encode(
                        DOMAIN_TYPEHASH,
                        keccak256(bytes(name())),
                        getChainId(),
                        address(this)
                )
        );

        bytes32 structHash = keccak256(
                abi.encode(
                        DELEGATION_TYPEHASH,
                        delegatee,
                        nonce,
                        expiry
                )
        );

        bytes32 digest = keccak256(
                abi.encodePacked(
                        "\x19\x01",
                        domainSeparator,
                        structHash
                )
        );

        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "MdxToken::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "MdxToken::delegateBySig: invalid nonce");
        require(now <= expiry, "MdxToken::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
}

/**
  * @notice Gets the current votes balance for `account`
  * @param account The address to get votes balance
  * @return The number of current votes for `account`
  */
function getCurrentVotes(address account)
external
view
returns (uint256)
{
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
}

/**
  * @notice Determine the prior number of votes for an account as of a block number
  * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
  * @param account The address of the account to check
  * @param blockNumber The block number to get the vote balance at
  * @return The number of votes the account had as of the given block
  */
```

```
    function getPriorVotes(address account, uint blockNumber)
    external
    view
    returns (uint256)
    {
        require(blockNumber < block.number, "MdxToken::getPriorVotes: not yet determined");

        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
            return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }

    function _delegate(address delegator, address delegatee)
    internal
    {
        address currentDelegate = _delegates[delegator];
        uint256 delegatorBalance = balanceOf(delegator); // balance of underlying balances (not scaled);
        _delegates[delegator] = delegatee;

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);

        emit DelegateChanged(delegator, currentDelegate, delegatee);
    }

    function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                // decrease old representative
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint256 srcRepNew = srcRepOld.sub(amount);
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
            }

            if (dstRep != address(0)) {
                // increase new representative
                uint32 dstRepNum = numCheckpoints[dstRep];
                uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                uint256 dstRepNew = dstRepOld.add(amount);
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
            }
        }
    }

    function _writeCheckpoint(
        address delegatee,
        uint32 nCheckpoints,
        uint256 oldVotes,
        uint256 newVotes
    )
    internal
    {
        uint32 blockNumber = safe32(block.number, "MdxToken::_writeCheckpoint: block number exceeds 32
bits");

        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
            checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
        } else {
            checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
            numCheckpoints[delegatee] = nCheckpoints + 1;
        }
```

```
        emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
    }

    function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
        require(n < 2**32, errorMessage);
        return uint32(n);
    }

    function getChainId() internal pure returns (uint) {
        uint256 chainId;
        assembly { chainId := chainid() }

        return chainId;
    }

    /// @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

    /// @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

}

contract MdxToken is DelegateERC20, Ownable {// knownsec MdxToken 合约
    uint256 private constant preMineSupply = 100000000 * 1e18; // pre-mine
    uint256 private constant maxSupply = 1000000000 * 1e18;     // the total supply

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _minters;

    constructor() public ERC20("MDX Token", "MDX"){
        _mint(msg.sender, preMineSupply);// knownsec 构造函数 铸币
    }

    // mint with max supply
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {// knownsec Minter 可用 铸币
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyOwner returns (bool) {// knownsec Owner 可用 Minter 增加
        require(_addMinter != address(0), "MdxToken: _addMinter is the zero address");
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyOwner returns (bool) {// knownsec Owner 可用 Minter 删除
        require(_delMinter != address(0), "MdxToken: _delMinter is the zero address");
        return EnumerableSet.remove(_minters, _delMinter);
    }

    function getMinterLength() public view returns (uint256) {// knownsec 获取矿工长度
        return EnumerableSet.length(_minters);
    }

    function isMinter(address account) public view returns (bool) {// knownsec 矿工资格查询
        return EnumerableSet.contains(_minters, account);
    }

    function getMinter(uint256 _index) public view onlyOwner returns (address){// knownsec Owner 可用 矿工获取，通过 index
        require(_index <= getMinterLength() - 1, "MdxToken: index out of bounds");
        return EnumerableSet.at(_minters, _index);
    }

    // modifier for mint function
    modifier onlyMinter() {// knownsec Minter 可用修饰器
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }

}
```

Router.sol

```
pragma solidity =0.6.6;

import "@openzeppelin/contracts/access/Ownable.sol";
import "../library/SafeMath.sol";
import "../interface/IERC20.sol";
```

```
import "../interface/IMdexFactory.sol";
import "../interface/IMdexPair.sol";

interface IMdexRouter {
    function factory() external pure returns (address);

    function WHT() external pure returns (address);

    function swapMining() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);

    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);

    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);

    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH);

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
```

```
        external
        payable
        returns (uint[] memory amounts);

        function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
        external
        returns (uint[] memory amounts);

        function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        returns (uint[] memory amounts);

        function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
        external
        payable
        returns (uint[] memory amounts);

        function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) external view returns (uint256 amountB);

        function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) external view returns (uint256 amountOut);

        function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) external view returns (uint256 amountIn);

        function getAmountsOut(uint256 amountIn, address[] calldata path) external view returns (uint256[] memory amounts);

        function getAmountsIn(uint256 amountOut, address[] calldata path) external view returns (uint256[] memory amounts);

        function removeLiquidityETHSupportingFeeOnTransferTokens(
            address token,
            uint liquidity,
            uint amountTokenMin,
            uint amountETHMin,
            address to,
            uint deadline
        ) external returns (uint amountETH);

        function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
            address token,
            uint liquidity,
            uint amountTokenMin,
            uint amountETHMin,
            address to,
            uint deadline,
            bool approveMax, uint8 v, bytes32 r, bytes32 s
        ) external returns (uint amountETH);

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

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

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

interface ISwapMining {
        function swap(address account, address input, address output, uint256 amount) external returns (bool);
}

interface IWHT {
        function deposit() external payable;

        function transfer(address to, uint value) external returns (bool);

        function withdraw(uint) external;
```

```
}

contract MdexRouter is IMdexRouter, Ownable {// knownsec Mdex 路由合约
    using SafeMath for uint256;

    address public immutable override factory;
    address public immutable override WHT;
    address public override swapMining;

    modifier ensure(uint deadline) {// knownsec  未达到 deadline 执行修饰符
        require(deadline >= block.timestamp, 'MdexRouter: EXPIRED');
        _;
    }

    constructor(address _factory, address _WHT) public {
        factory = _factory;
        WHT = _WHT;
    }// knownsec 构造函数，设置_factory 和_WHT

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

    function pairFor(address tokenA, address tokenB) public view returns (address pair){
        pair = IMdexFactory(factory).pairFor(tokenA, tokenB);
    }

    function setSwapMining(address _swapMininng) public onlyOwner {// knownsec Owner 可用  swapMining 设
置
        swapMining = _swapMininng;
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {// knownsec  增加流动性
        // create the pair if it doesn't exist yet
        if (IMdexFactory(factory).getPair(tokenA, tokenB) == address(0)) {// knownsec  创建交易对
            IMdexFactory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = IMdexFactory(factory).getReserves(tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = IMdexFactory(factory).quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'MdexRouter: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = IMdexFactory(factory).quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'MdexRouter: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin,
amountBMin);
        address pair = pairFor(tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IMdexPair(pair).mint(to);
    }

    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
```

```
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountETH, uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WHT,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = pairFor(token, WHT);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWHT(WHT).deposit{value : amountETH}();
        assert(IWHT(WHT).transfer(pair, amountETH));
        liquidity = IMdexPair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {// knownsec 移除流动性
        address pair = pairFor(tokenA, tokenB);
        IMdexPair(pair).transferFrom(msg.sender, pair, liquidity);
        // send liquidity to pair
        (uint amount0, uint amount1) = IMdexPair(pair).burn(to);
        (address token0,) = IMdexFactory(factory).sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'MdexRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'MdexRouter: INSUFFICIENT_B_AMOUNT');
    }

    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WHT,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWHT(WHT).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountA, uint amountB) {// knownsec 使用许可移除流动性
        address pair = pairFor(tokenA, tokenB);
        uint value = approveMax ? uint(- 1) : liquidity;
        IMdexPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin, to, deadline);
    }

    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
```

```
    ) external virtual override returns (uint amountToken, uint amountETH) {
        address pair = pairFor(token, WHT);
        uint value = approveMax ? uint(- 1) : liquidity;
        IMdexPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin, amountETHMin,
to, deadline);
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WHT,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWHT(WHT).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountETH) {
        address pair = pairFor(token, WHT);
        uint value = approveMax ? uint(- 1) : liquidity;
        IMdexPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = IMdexFactory(factory).sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOut);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut,
uint(0));
            address to = i < path.length - 2 ? pairFor(output, path[i + 2]) : _to;
            IMdexPair(pairFor(input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = IMdexFactory(factory).getAmountsOut(amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'MdexRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }

    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
```

```
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = IMdexFactory(factory).getAmountsIn(amountOut, path);
        require(amounts[0] <= amountInMax, 'MdexRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }

    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    virtual
    override
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[0] == WHT, 'MdexRouter: INVALID_PATH');
        amounts = IMdexFactory(factory).getAmountsOut(msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'MdexRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWHT(WHT).deposit{value : amounts[0]}();
        assert(IWHT(WHT).transfer(pairFor(path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
    }

    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to,
uint deadline)
    external
    virtual
    override
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WHT, 'MdexRouter: INVALID_PATH');
        amounts = IMdexFactory(factory).getAmountsIn(amountOut, path);
        require(amounts[0] <= amountInMax, 'MdexRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWHT(WHT).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to,
uint deadline)
    external
    virtual
    override
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WHT, 'MdexRouter: INVALID_PATH');
        amounts = IMdexFactory(factory).getAmountsOut(amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'MdexRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWHT(WHT).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
    external
    virtual
    override
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[0] == WHT, 'MdexRouter: INVALID_PATH');
        amounts = IMdexFactory(factory).getAmountsIn(amountOut, path);
        require(amounts[0] <= msg.value, 'MdexRouter: EXCESSIVE_INPUT_AMOUNT');
        IWHT(WHT).deposit{value : amounts[0]}();
        assert(IWHT(WHT).transfer(pairFor(path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]);
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
```

```
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = IMdexFactory(factory).sortTokens(input, output);
            IMdexPair pair = IMdexPair(pairFor(input, output));
            uint amountInput;
            uint amountOutput;
            {// scope to avoid stack too deep errors
                (uint reserve0, uint reserve1,) = pair.getReserves();
                (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
                amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
                amountOutput = IMdexFactory(factory).getAmountOut(amountInput, reserveInput, reserveOutput);
            }
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOutput);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amountOutput, uint(0));
            address to = i < path.length - 2 ? pairFor(output, path[i + 2]) : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amountIn
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'MdexRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
    external
    virtual
    override
    payable
    ensure(deadline)
    {
        require(path[0] == WHT, 'MdexRouter: INVALID_PATH');
        uint amountIn = msg.value;
        IWHT(WHT).deposit{value : amountIn}();
        assert(IWHT(WHT).transfer(pairFor(path[0], path[1]), amountIn));
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'MdexRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
    external
    virtual
    override
    ensure(deadline)
    {
        require(path[path.length - 1] == WHT, 'MdexRouter: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WHT).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'MdexRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        IWHT(WHT).withdraw(amountOut);
```

```
        TransferHelper.safeTransferETH(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) public view override returns (uint256 amountB) {
        return IMdexFactory(factory).quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) public view override returns (uint256 amountOut){
        return IMdexFactory(factory).getAmountOut(amountIn, reserveIn, reserveOut);
    }

    function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) public view override returns (uint256 amountIn){
        return IMdexFactory(factory).getAmountIn(amountOut, reserveIn, reserveOut);
    }

    function getAmountsOut(uint256 amountIn, address[] memory path) public view override returns (uint256[] memory amounts){
        return IMdexFactory(factory).getAmountsOut(amountIn, path);
    }

    function getAmountsIn(uint256 amountOut, address[] memory path) public view override returns (uint256[] memory amounts){
        return IMdexFactory(factory).getAmountsIn(amountOut, path);
    }

}


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

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

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

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

SwapMining.sol

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

pragma solidity >=0.6.0 <0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "../interface/IERC20.sol";
import "../library/SafeMath.sol";
import "../interface/IMdexFactory.sol";
import "../interface/IMdexPair.sol";
import "../interface/IMdx.sol";

interface IOracle {
    function update(address tokenA, address tokenB) external;

    function consult(address tokenIn, uint amountIn, address tokenOut) external view returns (uint amountOut);
}

contract SwapMining is Ownable {// knownsec SwapMining 合约
    using SafeMath for uint256;// knownsec  安全函数库使用声明
    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _whitelist;
```

```
    // MDX tokens created per block
    uint256 public mdxPerBlock;// knownsec 初始化
    // The block number when MDX mining starts.
    uint256 public startBlock;
    // How many blocks are halved
    uint256 public halvingPeriod = 14400;
    // Total allocation points
    uint256 public totalAllocPoint = 0;
    IOracle public oracle;
    // router address
    address public router;
    // factory address
    IMdexFactory public factory;
    // mdx token address
    IMdx public mdx;
    // Calculate price based on HUSD
    address public targetToken;
    // pair corresponding pid
    mapping(address => uint256) public pairOfPid;

    constructor(
        IMdx _mdx,
        IMdexFactory _factory,
        IOracle _oracle,
        address _router,
        address _targetToken,
        uint256 _mdxPerBlock,
        uint256 _startBlock
    ) public {// knownsec 构造函数 传入 _mdx _factory _oracle _router _targetToken _mdxPerBlock
_startBlock
        mdx = _mdx;
        factory = _factory;
        oracle = _oracle;
        router = _router;
        targetToken = _targetToken;
        mdxPerBlock = _mdxPerBlock;
        startBlock = _startBlock;
    }

    struct UserInfo {// knownsec 用户信息结构体
        uint256 quantity;     // How many LP tokens the user has provided
        uint256 blockNumber;  // Last transaction block
    }

    struct PoolInfo {// knownsec 池信息结构体
        address pair;           // Trading pairs that can be mined
        uint256 quantity;       // Current amount of LPs
        uint256 totalQuantity;  // All quantity
        uint256 allocPoint;     // How many allocation points assigned to this pool
        uint256 allocMdxAmount; // How many MDXs
        uint256 lastRewardBlock;// Last transaction block
    }

    PoolInfo[] public poolInfo;// knownsec 池信息数组
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;// knownsec 用户信息映射


    function poolLength() public view returns (uint256) {// knownsec 池长度获取
        return poolInfo.length;
    }


    function addPair(uint256 _allocPoint, address _pair, bool _withUpdate) public onlyOwner {// knownsec
Owner 可用 增加池
        if (_withUpdate) {
            massMintPools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
        pair : _pair,
        quantity : 0,
        totalQuantity : 0,
        allocPoint : _allocPoint,
        allocMdxAmount : 0,
        lastRewardBlock : lastRewardBlock
        }));
        pairOfPid[_pair] = poolLength() - 1;
    }

    // Update the allocPoint of the pool
    function setPair(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {// knownsec Owner
可用 设置对应池的 allocPoint
        if (_withUpdate) {
            massMintPools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
```

```
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Set the number of mdx produced by each block
    function setMdxPerBlock(uint256 _newPerBlock) public onlyOwner {// knownsec Owner 可用 设置 mdxPerBlock
        massMintPools();
        mdxPerBlock = _newPerBlock;
    }

    // Only tokens in the whitelist can be mined MDX
    function addWhitelist(address _addToken) public onlyOwner returns (bool) {// knownsec Owner 可用 增加 MDX 挖掘白名单
        require(_addToken != address(0), "SwapMining: token is the zero address");
        return EnumerableSet.add(_whitelist, _addToken);
    }

    function delWhitelist(address _delToken) public onlyOwner returns (bool) {// knownsec Owner 可用 删除 MDX 挖掘白名单
        require(_delToken != address(0), "SwapMining: token is the zero address");
        return EnumerableSet.remove(_whitelist, _delToken);
    }

    function getWhitelistLength() public view returns (uint256) {// knownsec 白名单长度查看
        return EnumerableSet.length(_whitelist);
    }

    function isWhitelist(address _token) public view returns (bool) {// knownsec 白名单查询
        return EnumerableSet.contains(_whitelist, _token);
    }

    function getWhitelist(uint256 _index) public view returns (address) {// knownsec 获取对应 index 白名单
        require(_index <= getWhitelistLength() - 1, "SwapMining: index out of bounds");
        return EnumerableSet.at(_whitelist, _index);
    }

    function setHalvingPeriod(uint256 _block) public onlyOwner {// knownsec Owner 可用 减半周期设置
        halvingPeriod = _block;
    }

    function setRouter(address newRouter) public onlyOwner {// knownsec Owner 可用 路由合约设置
        require(newRouter != address(0), "SwapMining: new router is the zero address");
        router = newRouter;
    }

    function setOracle(IOracle _oracle) public onlyOwner {// knownsec Owner 可用 预言机合约设置
        require(_oracle != address(0), "SwapMining: new oracle is the zero address");
        oracle = _oracle;
    }

    // At what phase
    function phase(uint256 blockNumber) public view returns (uint256) {// knownsec 周期段检测
        if (halvingPeriod == 0) {
            return 0;
        }
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function phase() public view returns (uint256) {
        return phase(block.number);
    }

    function reward(uint256 blockNumber) public view returns (uint256) {// knownsec Owner 可用 周期奖励查看
        uint256 _phase = phase(blockNumber);
        return mdxPerBlock.div(2 ** _phase);
    }

    function reward() public view returns (uint256) {
        return reward(block.number);
    }

    // Rewards for the current block
    function getMdxReward(uint256 _lastRewardBlock) public view returns (uint256) {// knownsec 所处 MdxBlock 奖励阶段计算 公开 view
        uint256 blockReward = 0;
        uint256 n = phase(_lastRewardBlock);
        uint256 m = phase(block.number);
        // If it crosses the cycle
        while (n < m) {
            n++;
            // Get the last block of the previous cycle
            uint256 r = n.mul(halvingPeriod).add(startBlock);
            // Get rewards from previous periods
```

```
                blockReward = blockReward.add((r.sub(_lastRewardBlock)).mul(reward(r)));
                _lastRewardBlock = r;
            }
            blockReward = blockReward.add((block.number.sub(_lastRewardBlock)).mul(reward(block.number)));
            return blockReward;
        }

        // Update all pools Called when updating allocPoint and setting new blocks
        function massMintPools() public {// knownsec 更新 allocPoint 和设置新块时调用的所有池 公开
            uint256 length = poolInfo.length;
            for (uint256 pid = 0; pid < length; ++pid) {// knownsec 池信息遍历
                mint(pid);
            }
        }

        function mint(uint256 _pid) public returns (bool) {
            PoolInfo storage pool = poolInfo[_pid];// knownsec 取出池结构体
            if (block.number <= pool.lastRewardBlock) {
                return false;
            }
            uint256 blockReward = getMdxReward(pool.lastRewardBlock);
            if (blockReward <= 0) {
                return false;
            }
            // Calculate the rewards obtained by the pool based on the allocPoint
            uint256 mdxReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
            mdx.mint(address(this), mdxReward);// knownsec 奖励铸币到合约本身
            // Increase the number of tokens in the current pool
            pool.allocMdxAmount = pool.allocMdxAmount.add(mdxReward);
            pool.lastRewardBlock = block.number;
            return true;
        }

        // swapMining only router
        function swap(address account, address input, address output, uint256 amount) public onlyRouter returns
(bool) {// knownsec 路由合约可用 交换铸币
            require(account != address(0), "SwapMining: taker swap account is the zero address");
            require(input != address(0), "SwapMining: taker swap input is the zero address");
            require(output != address(0), "SwapMining: taker swap output is the zero address");

            if (poolLength() <= 0) {
                return false;
            }

            if (!isWhitelist(input) || !isWhitelist(output)) {
                return false;
            }

            address pair = IMdexFactory(factory).pairFor(input, output);

            PoolInfo storage pool = poolInfo[pairOfPid[pair]];
            // If it does not exist or the allocPoint is 0 then return
            if (pool.pair != pair || pool.allocPoint <= 0) {
                return false;
            }

            uint256 quantity = getQuantity(output, amount, targetToken);
            if (quantity <= 0) {
                return false;
            }

            mint(pairOfPid[pair]);

            pool.quantity = pool.quantity.add(quantity);
            pool.totalQuantity = pool.totalQuantity.add(quantity);
            UserInfo storage user = userInfo[pairOfPid[pair]][account];
            user.quantity = user.quantity.add(quantity);
            user.blockNumber = block.number;
            return true;
        }

        // The user withdraws all the transaction rewards of the pool
        function takerWithdraw() public {// knownsec 池中该用户所有交易奖励提取 公开
            uint256 userSub;
            uint256 length = poolInfo.length;
            for (uint256 pid = 0; pid < length; ++pid) {
                PoolInfo storage pool = poolInfo[pid];
                UserInfo storage user = userInfo[pid][msg.sender];
                if (user.quantity > 0) {
                    mint(pid);
                    // The reward held by the user in this pool
                    uint256 userReward = pool.allocMdxAmount.mul(user.quantity).div(pool.quantity);
                    pool.quantity = pool.quantity.sub(user.quantity);
                    pool.allocMdxAmount = pool.allocMdxAmount.sub(userReward);
                    user.quantity = 0;
                    user.blockNumber = block.number;
                    userSub = userSub.add(userReward);
```

```
                }
            }
            if (userSub <= 0) {
                return;
            }
            mdx.transfer(msg.sender, userSub);
        }

        // Get rewards from users in the current pool
        function getUserReward(uint256 _pid) public view returns (uint256, uint256){// knownsec 池中pid对应用户奖励获取
            require(_pid <= poolInfo.length - 1, "SwapMining: Not find this pool");
            uint256 userSub;
            PoolInfo memory pool = poolInfo[_pid];
            UserInfo memory user = userInfo[_pid][msg.sender];
            if (user.quantity > 0) {
                uint256 blockReward = getMdxReward(pool.lastRewardBlock);
                uint256 mdxReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                userSub = userSub.add((pool.allocMdxAmount.add(mdxReward)).mul(user.quantity).div(pool.quantity));
            }
            //Mdx available to users, User transaction amount
            return (userSub, user.quantity);
        }

        // Get details of the pool
        function getPoolInfo(uint256 _pid) public view returns (address, address, uint256, uint256, uint256){// knownsec 池信息获取
            require(_pid <= poolInfo.length - 1, "SwapMining: Not find this pool");
            PoolInfo memory pool = poolInfo[_pid];
            address token0 = IMdexPair(pool.pair).token0();
            address token1 = IMdexPair(pool.pair).token1();
            uint256 mdxAmount = pool.allocMdxAmount;
            uint256 blockReward = getMdxReward(pool.lastRewardBlock);
            uint256 mdxReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
            mdxAmount = mdxAmount.add(mdxReward);
            //token0,token1,Pool remaining reward,Total /Current transaction volume of the pool
            return (token0, token1, mdxAmount, pool.totalQuantity, pool.quantity, pool.allocPoint);
        }

        modifier onlyRouter() {// knownsec 路由合约可用修饰器
            require(msg.sender == router, "SwapMining: caller is not the router");
            _;
        }

        function getQuantity(address outputToken, uint256 outputAmount, address anchorToken) public view returns (uint256) {// knownsec 预言机价格获取 公开
            uint256 quantity = 0;
            if (outputToken == anchorToken) {
                quantity = outputAmount;
            } else if (IMdexFactory(factory).getPair(outputToken, anchorToken) != address(0)) {
                quantity = IOracle(oracle).consult(outputToken, outputAmount, anchorToken);
            } else {
                uint256 length = getWhitelistLength();
                for (uint256 index = 0; index < length; index++) {
                    address intermediate = getWhitelist(index);
                    if (IMdexFactory(factory).getPair(outputToken, intermediate) != address(0) && IMdexFactory(factory).getPair(intermediate, anchorToken) != address(0)) {
                        uint256 interQuantity = IOracle(oracle).consult(outputToken, outputAmount, intermediate);
                        quantity = IOracle(oracle).consult(intermediate, interQuantity, anchorToken);
                        break;
                    }
                }
            }
            return quantity;
        }

}
```

IERC20.sol

```
pragma solidity >=0.5.0 <0.8.0;

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

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view returns (uint8);

    function totalSupply() external view returns (uint);
```

```
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);
    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);
}
```

IMdexFactory.sol

```
pragma solidity >=0.5.0 <0.8.0;

interface IMdexFactory {/// knownsec IMdexFactory 接口
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);
    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);
    function feeToRate() external view returns (uint256);
    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);
    function createPair(address tokenA, address tokenB) external returns (address pair);
    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
    function setFeeToRate(uint256) external;
    function sortTokens(address tokenA, address tokenB) external pure returns (address token0, address token1);
    function pairFor(address tokenA, address tokenB) external view returns (address pair);
    function getReserves(address tokenA, address tokenB) external view returns (uint256 reserveA, uint256 reserveB);
    function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) external pure returns (uint256 amountB);
    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) external view returns (uint256 amountOut);
    function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) external view returns (uint256 amountIn);
    function getAmountsOut(uint256 amountIn, address[] calldata path) external view returns (uint256[] memory amounts);
    function getAmountsIn(uint256 amountOut, address[] calldata path) external view returns (uint256[] memory amounts);
}
```

IMdexPair.sol

```
pragma solidity >=0.5.0 <0.8.0;

interface IMdexPair {/// knownsec IMdexPair 接口
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);
    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);
    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
```

```
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);

    function PERMIT_TYPEHASH() external pure returns (bytes32);

    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;

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

    function MINIMUM_LIQUIDITY() external pure returns (uint);

    function factory() external view returns (address);

    function token0() external view returns (address);

    function token1() external view returns (address);

    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);

    function price0CumulativeLast() external view returns (uint);

    function price1CumulativeLast() external view returns (uint);

    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);

    function burn(address to) external returns (uint amount0, uint amount1);

    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;

    function skim(address to) external;

    function sync() external;

    function price(address token, uint256 baseDecimal) external view returns (uint256);

    function initialize(address, address) external;
}
```

IMdx.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import {IERC20 as SIERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IMdx is SIERC20 {// knownsec IMdx 接口
    function mint(address to, uint256 amount) external returns (bool);
}
```

SafeMath.sol

```
pragma solidity >=0.5.0 <0.8.0;

library SafeMath {// knownsec  安全数学库
    uint256 constant WAD = 10 ** 18;
    uint256 constant RAY = 10 ** 27;

    function wad() public pure returns (uint256) {
        return WAD;
    }

    function ray() public pure returns (uint256) {
        return RAY;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
```

```
        return c;
    }

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

    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    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;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        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;
    }

    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;
    }

    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a <= b ? a : b;
    }

    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    function sqrt(uint256 a) internal pure returns (uint256 b) {
        if (a > 3) {
            b = a;
            uint256 x = a / 2 + 1;
            while (x < b) {
                b = x;
                x = (a / x + x) / 2;
            }
        } else if (a != 0) {
            b = 1;
        }
    }

    function wmul(uint256 a, uint256 b) internal pure returns (uint256) {
        return mul(a, b) / WAD;
    }

    function wmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, b), WAD / 2) / WAD;
    }

    function rmul(uint256 a, uint256 b) internal pure returns (uint256) {
        return mul(a, b) / RAY;
    }

    function rmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, b), RAY / 2) / RAY;
    }
```

```
    function wdiv(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(mul(a, WAD), b);
    }

    function wdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, WAD), b / 2) / b;
    }

    function rdiv(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(mul(a, RAY), b);
    }

    function rdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, RAY), b / 2) / b;
    }

    function wpow(uint256 x, uint256 n) internal pure returns (uint256) {
        uint256 result = WAD;
        while (n > 0) {
            if (n % 2 != 0) {
                result = wmul(result, x);
            }
            x = wmul(x, x);
            n /= 2;
        }
        return result;
    }

    function rpow(uint256 x, uint256 n) internal pure returns (uint256) {
        uint256 result = RAY;
        while (n > 0) {
            if (n % 2 != 0) {
                result = rmul(result, x);
            }
            x = rmul(x, x);
            n /= 2;
        }
        return result;
    }
}
```

CoinChef.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "../interface/IMdx.sol";

interface IMasterChef {// knownsec IMasterChef 接口
    function pendingSushi(uint256 pid, address user) external view returns (uint256);

    function deposit(uint256 pid, uint256 amount) external;

    function withdraw(uint256 pid, uint256 amount) external;

    function emergencyWithdraw(uint256 pid) external;
}

contract CoinChef is Ownable {// knownsec CoinChef 合约
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _sushiLP;

    // Info of each user.
    struct UserInfo {// knownsec  用户信息结构体
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt.
        uint256 sushiRewardDebt; //sushi Reward debt.
    }

    // Info of each pool.
    struct PoolInfo {// knownsec  池信息结构体
        IERC20 lpToken;           // Address of LP token contract.
        uint256 allocPoint;             // How many allocation points assigned to this pool. MDXs to distribute per
block.
        uint256 lastRewardBlock;    // Last block number that MDXs distribution occurs.
        uint256 accMdxPerShare; // Accumulated MDXs per share, times 1e12.
        uint256 totalAmount;        // Total amount of current pool deposit.
```

```
        uint256 accSushiPerShare; //Accumulated SuSHIs per share
    }

    // The MDX TOKEN!
    IMdx public mdx;
    // MDX tokens created per block.
    uint256 public constant mdxPerBlock = 100 ** 1e18;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
    // Corresponding to the pid of the sushi pool
    mapping(uint256 => uint256) public poolCorrespond;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when MDX mining starts.
    uint256 public startBlock;
    // The block number when MDX mining end;
    uint256 public endBlock;
    // SUSHI MasterChef 0xc2EdaD668740f1aA35E4D8f227fB8E17dcA888Cd
    address public constant sushiChef = 0xc2EdaD668740f1aA35E4D8f227fB8E17dcA888Cd;
    // SUSHI Token 0x6B3595068778DD592e39A122f4f5a5cF09C90fE2
    address public constant sushiToken = 0x6B3595068778DD592e39A122f4f5a5cF09C90fE2;

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

    constructor(
        IMdx _mdx,
        uint256 _startBlock
    ) public {
        mdx = _mdx;
        startBlock = _startBlock;
        endBlock = _startBlock.add(200000);
    }// knownsec   构造器 放入_mdx 和开始块

    function poolLength() public view returns (uint256) {
        return poolInfo.length;
    }

    function addSushiLP(address _addLP) public onlyOwner returns (bool) {// knownsec  增加 EnumerableSet 数组 owner 可用
        require(_addLP != address(0), "LP is the zero address");
        IERC20(_addLP).approve(sushiChef, uint256(- 1));
        return EnumerableSet.add(_sushiLP, _addLP);
    }

    function isSushiLP(address _LP) public view returns (bool) {
        return EnumerableSet.contains(_sushiLP, _LP);
    }

    function getSushiLPLength() public view returns (uint256) {
        return EnumerableSet.length(_sushiLP);
    }

    function getSushiLPAddress(uint256 _pid) public view returns (address){
        require(_pid <= getSushiLPLength() - 1, "not find this SushiLP");
        return EnumerableSet.at(_sushiLP, _pid);
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {// knownsec Owner 可用   增加新的 lp 到池中
        require(address(_lpToken) != address(0), "lpToken is the zero address");
        require(block.number < endBlock, "All token mining completed");
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
        lpToken : _lpToken,
        allocPoint : _allocPoint,
        lastRewardBlock : lastRewardBlock,
        accMdxPerShare : 0,
        totalAmount : 0,
        accSushiPerShare : 0
        }));
    }

    // Update the given pool's MDX allocation point. Can only be called by the owner.
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {// knownsec Owner 可用   更新 MDX 分配
        if (_withUpdate) {
            massUpdatePools();
```

```
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // The current pool corresponds to the pid of the sushi pool
    function setPoolCorr(uint256 _pid, uint256 _sid) public onlyOwner {// knownsec Owner 可用    设置池对应的 sid
        require(_pid <= poolLength() - 1, "not find this pool");
        poolCorrespond[_pid] = _sid;
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools() public {// knownsec    更新所有池的奖励
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {// knownsec    更新对应池的奖励
        PoolInfo storage pool = poolInfo[_pid];
        uint256 number = block.number > endBlock ? endBlock : block.number;
        if (number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply;
        if (isSushiLP(address(pool.lpToken))) {
            if (pool.totalAmount == 0) {
                pool.lastRewardBlock = number;
                return;
            }
            lpSupply = pool.totalAmount;
        } else {
            lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardBlock = number;
                return;
            }
        }

        uint256 multiplier = number.sub(pool.lastRewardBlock);
        uint256 mdxReward = multiplier.mul(mdxPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
        bool minRet = mdx.mint(address(this), mdxReward);
        if (minRet) {
            pool.accMdxPerShare = pool.accMdxPerShare.add(mdxReward.mul(1e12).div(lpSupply));
        }
        pool.lastRewardBlock = number;
    }

    // View function to see pending MDXs on frontend.
    function pending(uint256 _pid, address _user) external view returns (uint256, uint256){// knownsec  查看之前未被处理的 MDX  外部调用
        PoolInfo storage pool = poolInfo[_pid];
        if (isSushiLP(address(pool.lpToken))) {
            (uint256 mdxAmount, uint256 sushiAmount) = pendingMdxAndSushi(_pid, _user);
            return (mdxAmount, sushiAmount);
        } else {
            uint256 mdxAmount = pendingMdx(_pid, _user);
            return (mdxAmount, 0);
        }
    }

    function pendingMdxAndSushi(uint256 _pid, address _user) private view returns (uint256, uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accMdxPerShare = pool.accMdxPerShare;
        uint256 accSushiPerShare = pool.accSushiPerShare;
        uint256 number = block.number > endBlock ? endBlock : block.number;
        if (user.amount > 0) {
            uint256 sushiPending = IMasterChef(sushiChef).pendingSushi(poolCorrespond[_pid], address(this));
            accSushiPerShare = accSushiPerShare.add(sushiPending.mul(1e12).div(pool.totalAmount));
            uint256 userPending = user.amount.mul(accSushiPerShare).div(1e12).sub(user.sushiRewardDebt);
            if (number > pool.lastRewardBlock) {
                uint256 multiplier = number.sub(pool.lastRewardBlock);
                uint256 mdxReward = multiplier.mul(mdxPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
                accMdxPerShare = accMdxPerShare.add(mdxReward.mul(1e12).div(pool.totalAmount));
                return (user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
            if (number == pool.lastRewardBlock) {
                return (user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
        }
        return (0, 0);
    }
```

```
    function pendingMdx(uint256 _pid, address _user) private view returns (uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accMdxPerShare = pool.accMdxPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        uint256 number = block.number > endBlock ? endBlock : block.number;
        if (user.amount > 0) {
            if (number > pool.lastRewardBlock) {
                uint256 multiplier = block.number.sub(pool.lastRewardBlock);
                uint256 mdxReward = multiplier.mul(mdxPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
                accMdxPerShare = accMdxPerShare.add(mdxReward.mul(1e12).div(lpSupply));
                return user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt);
            }
            if (number == pool.lastRewardBlock) {
                return user.amount.mul(accMdxPerShare).div(1e12).sub(user.rewardDebt);
            }
        }
        return 0;
    }

    // Deposit LP tokens to CoinChef for MDX allocation.
    function deposit(uint256 _pid, uint256 _amount) public {// knownsec 质押 LP 到 CoinChef
        PoolInfo storage pool = poolInfo[_pid];
        if (isSushiLP(address(pool.lpToken))) {
            depositMdxAndSushi(_pid, _amount, msg.sender);
        } else {
            depositMdx(_pid, _amount, msg.sender);
        }
    }

    function depositMdxAndSushi(uint256 _pid, uint256 _amount, address _user) private {// knownsec 质押 Mdx 与 Sushi 私有
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);// knownsec 池更新
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
            uint256 beforeSushi = IERC20(sushiToken).balanceOf(address(this));// knownsec 起始值计算
            IMasterChef(sushiChef).deposit(poolCorrespond[_pid], 0);
            uint256 afterSushi = IERC20(sushiToken).balanceOf(address(this));
            pool.accSushiPerShare = pool.accSushiPerShare.add(afterSushi.sub(beforeSushi).mul(1e12).div(pool.totalAmount));
            uint256 sushiPending = user.amount.mul(pool.accSushiPerShare).div(1e12).sub(user.sushiRewardDebt);
            if (sushiPending > 0) {
                IERC20(sushiToken).safeTransfer(_user, sushiPending);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            if (pool.totalAmount == 0) {
                IMasterChef(sushiChef).deposit(poolCorrespond[_pid], _amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
                user.amount = user.amount.add(_amount);
            } else {
                uint256 beforeSushi = IERC20(sushiToken).balanceOf(address(this));
                IMasterChef(sushiChef).deposit(poolCorrespond[_pid], _amount);
                uint256 afterSushi = IERC20(sushiToken).balanceOf(address(this));
                pool.accSushiPerShare = pool.accSushiPerShare.add(afterSushi.sub(beforeSushi).mul(1e12).div(pool.totalAmount));
                pool.totalAmount = pool.totalAmount.add(_amount);
                user.amount = user.amount.add(_amount);
            }
        }
        user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
        user.sushiRewardDebt = user.amount.mul(pool.accSushiPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }

    function depositMdx(uint256 _pid, uint256 _amount, address _user) private {// knownsec MDX 质押 私有
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
```

```
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            }
            user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
            emit Deposit(_user, _pid, _amount);
        }

        // Withdraw LP tokens from CoinChef.
        function withdraw(uint256 _pid, uint256 _amount) public {// knownsec 公开 回撤池中的LPToken
            PoolInfo storage pool = poolInfo[_pid];
            if (isSushiLP(address(pool.lpToken))) {
                withdrawMdxAndSushi(_pid, _amount, msg.sender);
            } else {
                withdrawMdx(_pid, _amount, msg.sender);
            }
        }

        function withdrawMdxAndSushi(uint256 _pid, uint256 _amount, address _user) private {// knownsec 私有
回撤池中的Mdx和Sushi
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_user];
            require(user.amount >= _amount, "withdrawMdxAndSushi: not good");
            updatePool(_pid);
            uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
            if (_amount > 0) {
                uint256 beforeSushi = IERC20(sushiToken).balanceOf(address(this));
                IMasterChef(sushiChef).withdraw(poolCorrespond[_pid], _amount);
                uint256 afterSushi = IERC20(sushiToken).balanceOf(address(this));
                pool.accSushiPerShare                                                    =
pool.accSushiPerShare.add(afterSushi.sub(beforeSushi).mul(1e12).div(pool.totalAmount));
                uint256                                    sushiPending                  =
user.amount.mul(pool.accSushiPerShare).div(1e12).sub(user.sushiRewardDebt);
                if (sushiPending > 0) {
                    IERC20(sushiToken).safeTransfer(_user, sushiPending);
                }
                user.amount = user.amount.sub(_amount);
                pool.totalAmount = pool.totalAmount.sub(_amount);
                pool.lpToken.safeTransfer(_user, _amount);
            }
            user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
            user.sushiRewardDebt = user.amount.mul(pool.accSushiPerShare).div(1e12);
            emit Withdraw(_user, _pid, _amount);
        }

        function withdrawMdx(uint256 _pid, uint256 _amount, address _user) private {// knownsec 私有 回撤池中
的Mdx
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_user];
            require(user.amount >= _amount, "withdrawMdx: not good");
            updatePool(_pid);
            uint256 pendingAmount = user.amount.mul(pool.accMdxPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeMdxTransfer(_user, pendingAmount);
            }
            if (_amount > 0) {
                user.amount = user.amount.sub(_amount);
                pool.totalAmount = pool.totalAmount.sub(_amount);
                pool.lpToken.safeTransfer(_user, _amount);
            }
            user.rewardDebt = user.amount.mul(pool.accMdxPerShare).div(1e12);
            emit Withdraw(_user, _pid, _amount);
        }

        // Withdraw without caring about rewards. EMERGENCY ONLY.
        function emergencyWithdraw(uint256 _pid) public {// knownsec 公开 紧急情况 不计算收益回撤
            PoolInfo storage pool = poolInfo[_pid];
            if (isSushiLP(address(pool.lpToken))) {
                emergencyWithdrawMdxAndSushi(_pid, msg.sender);
            } else {
                emergencyWithdrawMdx(_pid, msg.sender);
            }
        }

        function emergencyWithdrawMdxAndSushi(uint256 _pid, address _user) private {// knownsec 私有 紧急情
况 不计算收益回撤Mdx和Sushi
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_user];
            uint256 amount = user.amount;
            uint256 beforeSushi = IERC20(sushiToken).balanceOf(address(this));
            IMasterChef(sushiChef).withdraw(poolCorrespond[_pid], amount);
            uint256 afterSushi = IERC20(sushiToken).balanceOf(address(this));
            pool.accSushiPerShare                                                    =
pool.accSushiPerShare.add(afterSushi.sub(beforeSushi).mul(1e12).div(pool.totalAmount));
            user.amount = 0;
```

```
            user.rewardDebt = 0;
            pool.lpToken.safeTransfer(_user, amount);
            pool.totalAmount = pool.totalAmount.sub(amount);
            emit EmergencyWithdraw(_user, _pid, amount);
        }

        function emergencyWithdrawMdx(uint256 _pid, address _user) private {// knownsec 私有 紧急情况 回撤MDX
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_user];
            uint256 amount = user.amount;
            user.amount = 0;
            user.rewardDebt = 0;
            pool.lpToken.safeTransfer(_user, amount);
            pool.totalAmount = pool.totalAmount.sub(amount);
            emit EmergencyWithdraw(_user, _pid, amount);
        }

        // Safe MDX transfer function, just in case if rounding error causes pool to not have enough MDXs.
        function safeMdxTransfer(address _to, uint256 _amount) internal {// knownsec MDX 转账函数，检测余额内部
            uint256 mdxBal = mdx.balanceOf(address(this));
            if (_amount > mdxBal) {
                mdx.transfer(_to, mdxBal);
            } else {
                mdx.transfer(_to, _amount);
            }
        }

}
```

MdxToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MdxToken is ERC20("MDX Token", "MDX"), Ownable {// knownsec MdxToken 合约
    uint256 private constant maxSupply = 20000000 * 1e18;        // the total supply
    address public minter;

    // mint with max supply
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {// knownsec onlyMinter 可用挖矿到_to 账户
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function setMinter(address _newMinter) external {// knownsec 外部 设置新的矿工，
        require(minter == address(0), "has set up");// 需要在 minter 未指向地址才可以设置，否则不予设置
        require(_newMinter != address(0), "is zero address");
        minter = _newMinter;
    }
    // modifier for mint function
    modifier onlyMinter() {// knownsec onlyMinter 修饰器
        require(msg.sender == minter, "caller is not the minter");
        _;
    }
}
```

Oracle.sol

```
pragma solidity =0.6.6;

import "../interface/IMdexFactory.sol";
import "../interface/IMdexPair.sol";

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

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

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

```
library FixedPoint {
    // range: [0, 2**112 - 1]
    // resolution: 1 / 2**112
    struct uq112x112 {
        uint224 _x;
    }

    // range: [0, 2**144 - 1]
    // resolution: 1 / 2**112
    struct uq144x112 {
        uint _x;
    }

    uint8 private constant RESOLUTION = 112;

    // encode a uint112 as a UQ112x112
    function encode(uint112 x) internal pure returns (uq112x112 memory) {
        return uq112x112(uint224(x) << RESOLUTION);
    }

    // encodes a uint144 as a UQ144x112
    function encode144(uint144 x) internal pure returns (uq144x112 memory) {
        return uq144x112(uint256(x) << RESOLUTION);
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function div(uq112x112 memory self, uint112 x) internal pure returns (uq112x112 memory) {
        require(x != 0, 'FixedPoint: DIV_BY_ZERO');
        return uq112x112(self._x / uint224(x));
    }

    // multiply a UQ112x112 by a uint, returning a UQ144x112
    // reverts on overflow
    function mul(uq112x112 memory self, uint y) internal pure returns (uq144x112 memory) {
        uint z;
        require(y == 0 || (z = uint(self._x) * y) / y == uint(self._x), "FixedPoint:
MULTIPLICATION_OVERFLOW");
        return uq144x112(z);
    }

    // returns a UQ112x112 which represents the ratio of the numerator to the denominator
    // equivalent to encode(numerator).div(denominator)
    function fraction(uint112 numerator, uint112 denominator) internal pure returns (uq112x112 memory) {
        require(denominator > 0, "FixedPoint: DIV_BY_ZERO");
        return uq112x112((uint224(numerator) << RESOLUTION) / denominator);
    }

    // decode a UQ112x112 into a uint112 by truncating after the radix point
    function decode(uq112x112 memory self) internal pure returns (uint112) {
        return uint112(self._x >> RESOLUTION);
    }

    // decode a UQ144x112 into a uint144 by truncating after the radix point
    function decode144(uq144x112 memory self) internal pure returns (uint144) {
        return uint144(self._x >> RESOLUTION);
    }
}

library MdexOracleLibrary {// knownsec Mdex 预言机库
    using FixedPoint for *;

    // helper function that returns the current block timestamp within the range of uint32, i.e. [0, 2**32 - 1]
    function currentBlockTimestamp() internal view returns (uint32) {
        return uint32(block.timestamp % 2 ** 32);
    }

    // produces the cumulative price using counterfactuals to save gas and avoid a call to sync.
    function currentCumulativePrices(
        address pair
    ) internal view returns (uint price0Cumulative, uint price1Cumulative, uint32 blockTimestamp) {
        blockTimestamp = currentBlockTimestamp();
        price0Cumulative = IMdexPair(pair).price0CumulativeLast();
        price1Cumulative = IMdexPair(pair).price1CumulativeLast();

        // if time has elapsed since the last update on the pair, mock the accumulated price values
        (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast) = IMdexPair(pair).getReserves();
        if (blockTimestampLast != blockTimestamp) {
            // subtraction overflow is desired
            uint32 timeElapsed = blockTimestamp - blockTimestampLast;
            // addition overflow is desired
            // counterfactual
            price0Cumulative += uint(FixedPoint.fraction(reserve1, reserve0)._x) * timeElapsed;
            // counterfactual
            price1Cumulative += uint(FixedPoint.fraction(reserve0, reserve1)._x) * timeElapsed;
        }
    }
}
```

```
contract Oracle {// knownsec  预言机合约
    using FixedPoint for *;
    using SafeMath for uint;

    struct Observation {// knownsec Observation 结构体
        uint timestamp;
        uint price0Cumulative;
        uint price1Cumulative;
    }

    address public immutable factory;
    uint public constant CYCLE = 30 minutes;

    // mapping from pair address to a list of price observations of that pair
    mapping(address => Observation) public pairObservations;

    constructor(address factory_) public {
        factory = factory_;
    }


    function update(address tokenA, address tokenB) external {// knownsec  外部  更新配对合约 tokenA 和
tokenB
        address pair = IMdexFactory(factory).pairFor(tokenA, tokenB);

        Observation storage observation = pairObservations[pair];
        uint timeElapsed = block.timestamp - observation.timestamp;
        require(timeElapsed >= CYCLE, 'MDEXOracle: PERIOD_NOT_ELAPSED');
        (uint price0Cumulative, uint price1Cumulative,) = MdexOracleLibrary.currentCumulativePrices(pair);
        observation.timestamp = block.timestamp;
        observation.price0Cumulative = price0Cumulative;
        observation.price1Cumulative = price1Cumulative;
    }


    function computeAmountOut(
        uint priceCumulativeStart, uint priceCumulativeEnd,
        uint timeElapsed, uint amountIn
    ) private pure returns (uint amountOut) {
        // overflow is desired.
        FixedPoint.uq112x112 memory priceAverage = FixedPoint.uq112x112(
            uint224((priceCumulativeEnd - priceCumulativeStart) / timeElapsed)
        );
        amountOut = priceAverage.mul(amountIn).decode144();
    }


    function consult(address tokenIn, uint amountIn, address tokenOut) external view returns (uint amountOut)
{// knownsec  预言机外部询价
        address pair = IMdexFactory(factory).pairFor(tokenIn, tokenOut);
        Observation storage observation = pairObservations[pair];
        uint timeElapsed = block.timestamp - observation.timestamp;
        (uint price0Cumulative, uint price1Cumulative,) = MdexOracleLibrary.currentCumulativePrices(pair);
        (address token0,) = IMdexFactory(factory).sortTokens(tokenIn, tokenOut);

        if (token0 == tokenIn) {
            return     computeAmountOut(observation.price0Cumulative,     price0Cumulative,     timeElapsed,
amountIn);
        } else {
            return     computeAmountOut(observation.price1Cumulative,     price1Cumulative,     timeElapsed,
amountIn);
        }
    }
}
```

6. Appendix B: Vulnerability risk rating criteria

Smart contract vulnerability rating standards	
Vulnerability rating	Vulnerability rating description
High-risk vulnerabilities	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.;
Mid-risk vulnerability	Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.;
Low-risk vulnerabilities	Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.

7. Appendix C: Introduction to vulnerability testing tools

7.1 Manticore

A Manticore is a symbolic execution tool for analyzing binary files and smart contracts. A Manticore consists of a symbolic Ethereum virtual machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of the Solarium body.It also incorporates Ethersplay, a Bit of Traits of Bits visual disassembler for EVM bytecode, for visual analysis. Like binaries, Manticore provides a simple command-line interface and a Python API for analyzing EVM bytecode.

7.2 Oyente

Oyente is a smart contract analysis tool that can be used to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and so on.More conveniently, Oyente's design is modular, so this allows power users to implement and insert their own inspection logic to check the custom properties in their contracts.

7.3 securify. Sh

Securify verifies the security issues common to Ethereum's smart contracts, such as unpredictability of trades and lack of input verification, while fully automated and analyzing all possible execution paths, and Securify has a specific language for identifying vulnerabilities that enables the securities to focus on current security and other reliability issues at all times.

7.4 Echidna

Echidna is a Haskell library designed for fuzzy testing EVM code.

7.5 MAIAN

MAIAN is an automated tool used to find holes in Ethereum's smart contracts. MAIAN processes the bytecode of the contract and tries to set up a series of transactions to find and confirm errors.

7.6 ethersplay

Ethersplay is an EVM disassembler that includes correlation analysis tools.

7.7 IDA - evm entry

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

7.8 want - ide

Remix is a browser-based compiler and IDE that allows users to build ethereum contracts and debug transactions using Solarium language.

7.9 KnownSec Penetration Tester kit

KnownSec penetration tester's toolkit, developed, collected and used by KnownSec penetration tester engineers, contains batch automated testing tools, self-developed tools, scripts or utilization tools, etc. dedicated to testers.

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