KNOWNSEC

Degree of secrets ★ | Project team open

Smart contract audit report

Security status

Security

★ ★ ★ ★ ★

Chief test Officer：

Version description

Document name	Document MD5	Document version
Moonswap Contract audit report	e311f92e817d88b2eb477fde9030e000	V1.0

Version Description

Reviser	Revision	Revision time	Confidentiality level
Knownsec cloud security service team	Moonswap Contract audit report	2020.09.22	Project team open

Knownsec's statement

Knownsec only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. Knownsec cannot determine the security status of its smart contract and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to knownsec by the information provider as of the time this report is issued. Knownsec's assumption: There is no missing, tampered, deleted or concealed information in the provided information. If the provided information is missing, tampered with, deleted, concealed or reflected in the actual situation, Knownsec shall not be liable for any losses and adverse effects caused thereby.

目录

1. Summary

The effective test time of this report is from September 21, 2020 to September 22, 2020. During this period, the security and standardization of the Mineable smart contract code will be audited and used as the statistical basis for the report.

In this test, we know that Chuangyu engineers have conducted a comprehensive analysis on common vulnerabilities of smart contracts (see Chapter 3), and found that there is a risk of logic design defects. Users can set the address as an empty address, which may lead to unexpected situations. However, due to the difficulty of utilization, users generally do not set the address as an empty address, so the comprehensive evaluation is passed.

The result of this smart contract security audit: passed

Since the test process is conducted in non production environment, all codes are the latest backup. The test process is communicated with relevant interface personnel, and relevant test operation is conducted under the condition of controllable operation risk, so as to avoid the production operation risk and code security risk in the test process.

Target information of this test:

Token name	**moonswap**
Token address	**https://github.com/moon-migration/moonswap-eth-contract**
Code type	**Token code**
Code language	**Solidity**
Contract address	➢ https://github.com/moon-migration/moonswap-eth-contract/blob/master/contracts/Migrator.sol ➢ https://github.com/moon-migration/moonswap-core/blob/master/contracts/UniswapV2Factory.sol ➢ https://github.com/moon-migration/moonswap-eth-contract/blob/master/contracts/uniswapv2/CrosschainPair.sol ➢ https://github.com/moon-migration/moonswap-eth-contract/

	blob/master/contracts/uniswapv2/CrosschainFactory.sol ➢ https://github.com/moon-migration/moonswap-core/blob/master/contracts/MigratorFactory.sol ➢ https://github.com/moon-migration/moonswap-core/blob/master/contracts/MigratorPair.sol ➢ https://github.com/moon-migration/moonswap-eth-contract/blob/master/contracts/MoonFund.sol ➢ https://github.com/moon-migration/moonswap-core/blob/master/contracts/ConfluxStar.sol

Contract document and hash:

Contract documents	MD5
Migrator.sol	035703e3fb403829094f52470ec932ae
UniswapV2Factory.sol	25c9372fcfb98d714c94c1da417f8442
CrosschainPair.sol	8f837c3764c8a1282219505f539f4703
CrosschainFactory.sol	c267bb3b7453fc8d115c900a7f40d042
MigratorFactory.sol	3d49b488c46b7a27147e9f5aa6324b24
MigratorPair.sol	5348a9d72bc3b22287f81a6a2e0b8d11
MoonFund.sol	1bcc007556b6d8436cbb2a7a845aab6e
ConfluxStar.sol	946f9fa44b0b33fb8f27f193f6d77f53

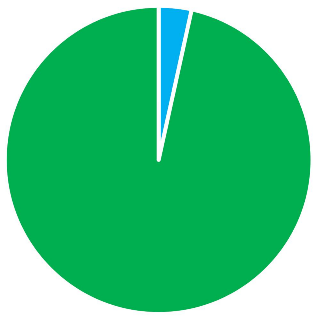
2. Code vulnerability analysis

2.1. Vulnerability level distribution

The risk of this vulnerability is counted by level:

Vulnerability risk level statistics table			
High risk	In danger	Low risk	Passed
0	0	1	28

Risk level distribution map



🟥 High risk[0个] 🟨 In danger[0个] 🟦 Low risk[1个] 🟩 Passed[28个]

2.2. Summary of audit results

Smart contract audit results			
Serial number	Test content	status	description
1	Reentry attack detection	Passed	After testing, there is no such safety problem.
2	Replay attack detection	Passed	After testing, there is no such safety problem.
3	Rearrangement attack detection	Passed	After testing, there is no such safety problem.
4	Numerical overflow detection	Passed	After testing, there is no such safety problem.
5	Arithmetic accuracy error	Passed	After testing, there is no such safety problem.
6	Access control defect detection	Passed	After testing, there is no such safety problem.
7	tx.progin authentication	Passed	After testing, there is no such safety problem.
8	call injection attack	Passed	After testing, there is no such safety problem.

9	Return value call verification	Passed	After testing, there is no such safety problem.
10	Uninitialized storage pointer	Passed	After testing, there is no such safety problem.
11	Wrong use of random number detection	Passed	After testing, there is no such safety problem.
12	Transaction order dependency detection	Passed	After testing, there is no such safety problem.
13	Denial of service attack detection	Passed	After testing, there is no such safety problem.
14	Logical design defect detection	Low risk （Passed）	After detection, there is a security problem. Users can set the address as an empty address, which may lead to unexpected situations. However, due to the difficulty of using, users generally do not set the address as an empty address
15	Fake recharge vulnerability detection	Passed	After testing, there is no such safety problem.
16	Additional token issuance vulnerability detection	Passed	After testing, there is no such safety problem.
17	Frozen account bypass detection	Passed	After testing, there is no such safety problem.
18	Compiler version security	Passed	After testing, there is no such safety problem.
19	Not recommended encoding	Passed	After testing, there is no such safety problem.
20	Redundant code	Passed	After testing, there is no such safety problem.
21	Use of safe arithmetic library	Passed	After testing, there is no such safety problem.
22	The use of require/assert use	Passed	After testing, there is no such safety problem.
23	gas consumption	Passed	After testing, there is no such safety problem.
24	Use of fallback function	Passed	After testing, there is no such safety problem.
25	owner permission control	Passed	After testing, there is no such safety problem.
26	Low-level function safety	Passed	After testing, there is no such safety problem.
27	Variable coverage	Passed	After testing, there is no such safety problem.
28	Timestamp dependent attack	Passed	After testing, there is no such safety problem.
29	Unsafe interface use	Passed	After testing, there is no such safety problem.

3. Analysis of code audit results

3.1. Reentry attack detection **[passed]**

Re-entry vulnerability is the most famous Ethereum smart contract vulnerability, which once led to the fork of Ethereum (TheDAO hack).

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

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

Safety advice: None.

3.2. Replay attack detection **[passed]**

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.

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

Safety advice: None.

3.3. Rearrangement attack detection **[passed]**

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping (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.

3.4. Numerical overflow detection **[passed]**

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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.5. Arithmetic accuracy error **[passed]**

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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.6. Access control detection [passed]

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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.7. tx.origin authentication [passed]

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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.8. call injection attack [passed]

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

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

Safety advice: None.

3.9. Return value call verification [passed]

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 Ether to an 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 coin transfer functions is not checked in the code, the contract will continue to execute the following code, which may cause unexpected results due to the failure of Ether sending.

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

Safety advice: None.

3.10. Uninitialized storage pointer [passed]

In solidity, a special data structure is allowed to be a struct structure, and local variables in a 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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.11. Wrong use of random numbers [passed]

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners, that is 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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.12. Transaction order depends on [passed]

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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.13. Denial of Service Attack [passed]

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, abuse of access control to access private components of the smart contract, use of confusion and negligence, etc. Wait.

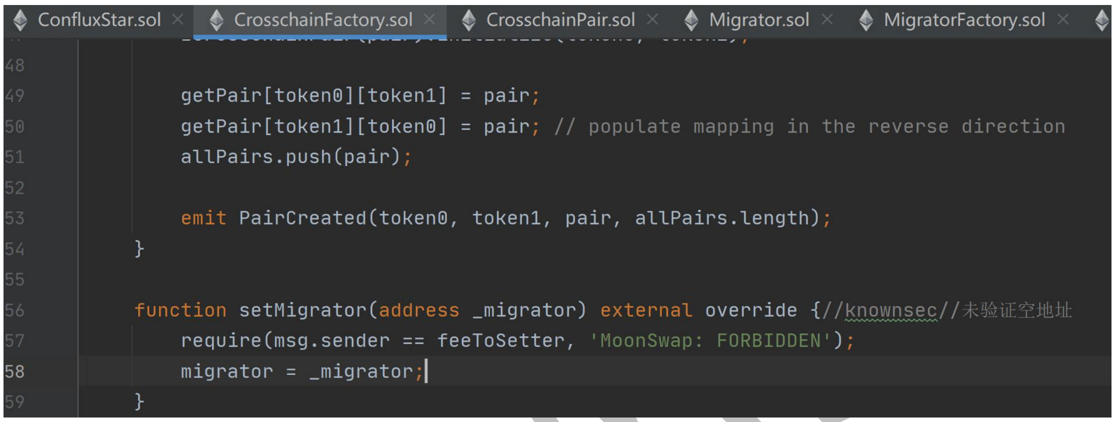
Test result: After testing, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

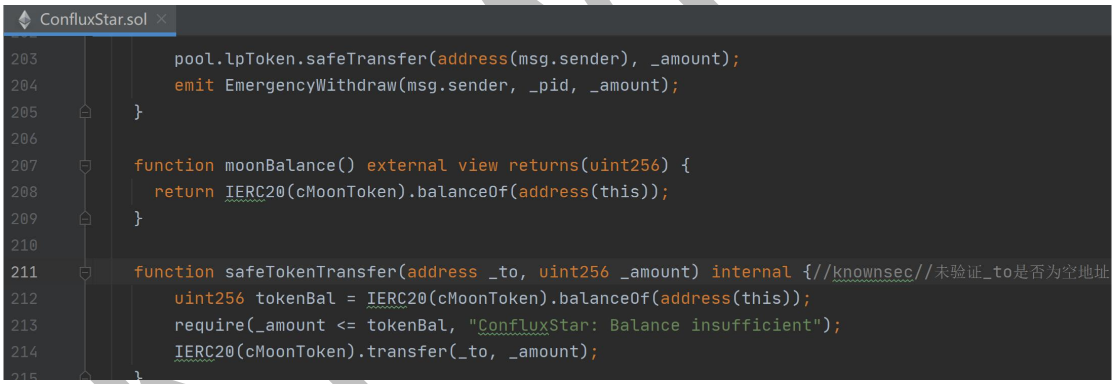
3.14. Logical design defect [Low risk]

Check the security issues related to business design in the smart contract code.

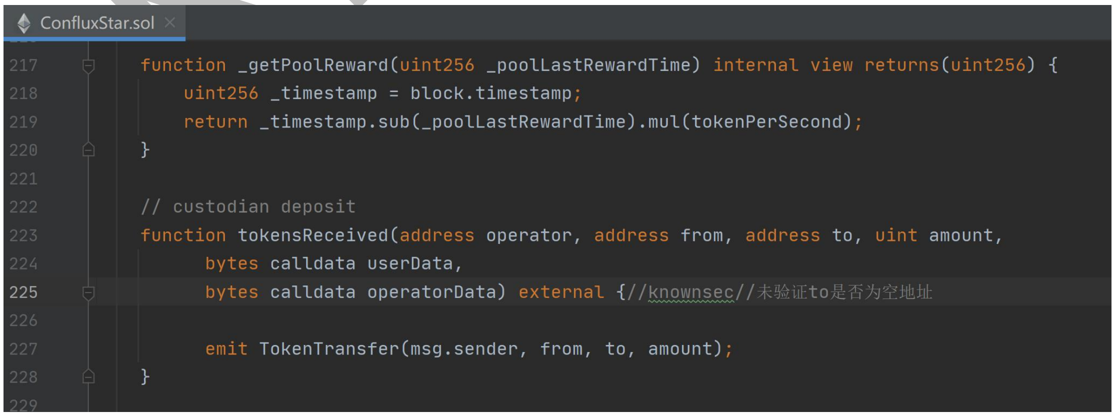
Test result: After detection, there is a risk of logic design defects in the setmigrator function and safetokenetransfer function in the contract. The user can set the address to an empty address, which may lead to unexpected situations. The code is as follows:



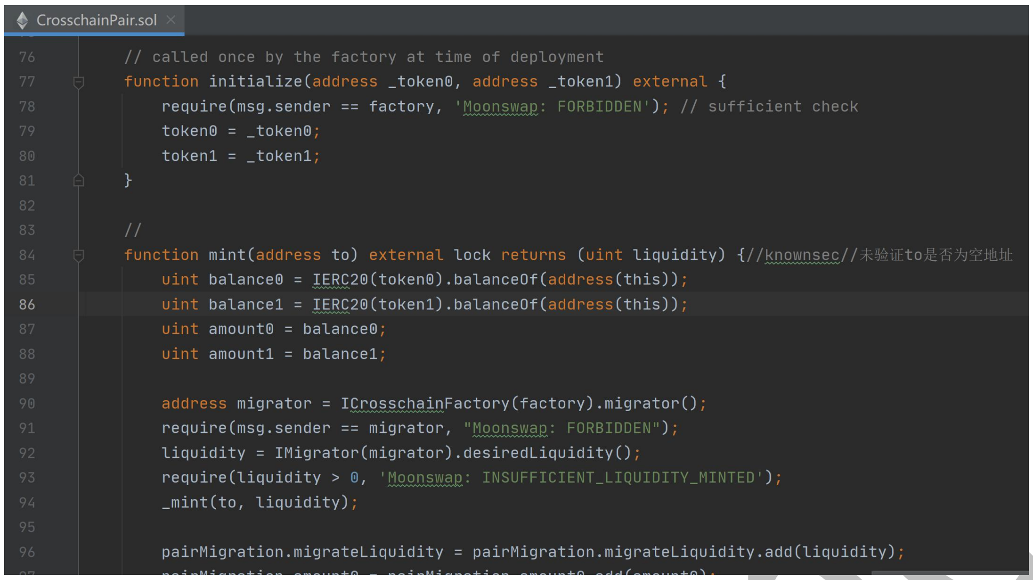
```
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);

        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setMigrator(address _migrator) external override {//knownsec//未验证空地址
        require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
        migrator = _migrator;
    }
```



```
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
        emit EmergencyWithdraw(msg.sender, _pid, _amount);
    }

    function moonBalance() external view returns(uint256) {
      return IERC20(cMoonToken).balanceOf(address(this));
    }

    function safeTokenTransfer(address _to, uint256 _amount) internal {//knownsec//未验证_to是否为空地址
        uint256 tokenBal = IERC20(cMoonToken).balanceOf(address(this));
        require(_amount <= tokenBal, "ConfluxStar: Balance insufficient");
        IERC20(cMoonToken).transfer(_to, _amount);
```

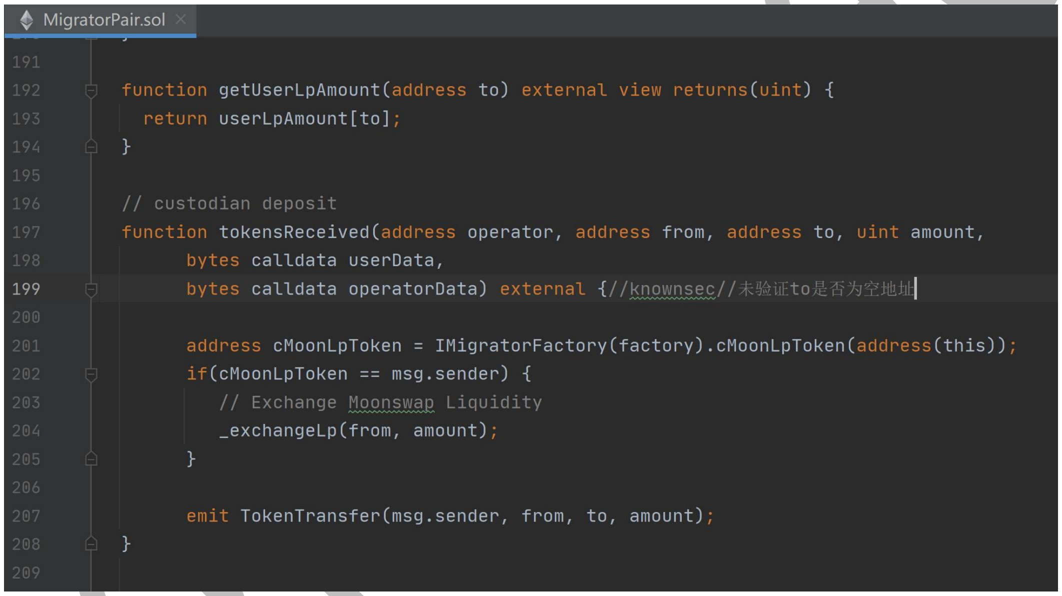


```
    function _getPoolReward(uint256 _poolLastRewardTime) internal view returns(uint256) {
        uint256 _timestamp = block.timestamp;
        return _timestamp.sub(_poolLastRewardTime).mul(tokenPerSecond);
    }

    // custodian deposit
    function tokensReceived(address operator, address from, address to, uint amount,
          bytes calldata userData,
          bytes calldata operatorData) external {//knownsec//未验证to是否为空地址

          emit TokenTransfer(msg.sender, from, to, amount);
    }
```



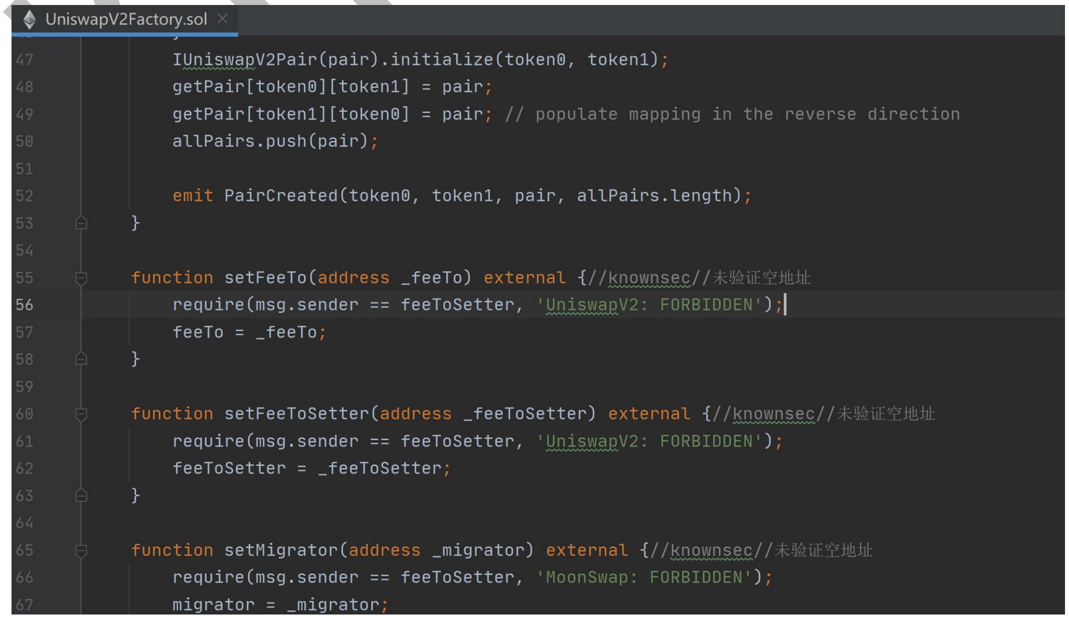
CrosschainPair.sol

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

    //
    function mint(address to) external lock returns (uint liquidity) {//knownsec//未验证to是否为空地址
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0;
        uint amount1 = balance1;

        address migrator = ICrosschainFactory(factory).migrator();
        require(msg.sender == migrator, "Moonswap: FORBIDDEN");
        liquidity = IMigrator(migrator).desiredLiquidity();
        require(liquidity > 0, 'Moonswap: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

        pairMigration.migrateLiquidity = pairMigration.migrateLiquidity.add(liquidity);
```



MigratorPair.sol

```

function getUserLpAmount(address to) external view returns(uint) {
  return userLpAmount[to];
}

// custodian deposit
function tokensReceived(address operator, address from, address to, uint amount,
      bytes calldata userData,
      bytes calldata operatorData) external {//knownsec//未验证to是否为空地址

      address cMoonLpToken = IMigratorFactory(factory).cMoonLpToken(address(this));
      if(cMoonLpToken == msg.sender) {
         // Exchange Moonswap Liquidity
         _exchangeLp(from, amount);
      }

      emit TokenTransfer(msg.sender, from, to, amount);
}

```



UniswapV2Factory.sol

```
        IUniswapV2Pair(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 {//knownsec//未验证空地址
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {//knownsec//未验证空地址
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }

    function setMigrator(address _migrator) external {//knownsec//未验证空地址
        require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
        migrator = _migrator;
```

Safety advice: Take the function setmigrator as an example to restrict the address from being empty. Add the following code:require(_ migrator != address(0));

3.15. Fake recharge loophole [passed]

The transfer function of the token contract uses if to check the balance of the transfer initiator (msg.sender). Judgment method, when balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else is a mild judgment method in sensitive function scenarios such as transfer. Not rigorous coding method.

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

Safety advice: None.

3.16. Vulnerabilities in additional token issuance [passed]

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, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

3.17. Freeze account bypass [passed]

In the source check token contract, when the token is transferred, whether there is an operation of whether the unverified token account, the originating account, and the target account are frozen.

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

Safety advice: None.

3.18. Compiler version security [passed]

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

Test results: after detection, the compiler version 0.5.8 or above is specified in the smart contract code, and there is no such security problem.

Safety advice: None.

3.19. Unrecommended encoding method [passed]

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

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

Safety advice: None.

3.20. Redundant code [passed]

Check whether the contract code implementation contains redundant code.

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

Safety advice: None.

3.21. The use of safe arithmetic library [passed]

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

Test results: after detection, the safemath security arithmetic library has been used in the smart contract code, and there is no security problem.

Safety advice: None.

3.22. Use of require/assert **[passed]**

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

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

Safety advice: None.

3.23. Gas consumption [passed]

Check whether the consumption of gas exceeds the maximum block limit.

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

Safety advice: None.

3.24. fallback function use [passed]

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

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

Safety advice: None.

3.25. owner permission control [passed]

Check whether the owner in the contract code implementation has excessive permissions. For example, arbitrarily modify the balance of other accounts.

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

Safety advice: None.

3.26. Low-level function security [passed]

Check whether there is a security hole in the use of call/delegatecall for the implementation of the contract code. The execution context of the call function is in the invoked contract; and the execution context of the delegatecall function is in the contract that currently calls the function.

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

Safety advice: None.

3.27. Variable coverage [passed]

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

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

Safety advice: None.

3.28. Timestamp dependency attack [passed]

The timestamp of data block usually uses the miner's local time, which can fluctuate in the range of 900 seconds. When other nodes accept a new block, they only need to verify whether the timestamp is later than the previous block and the error with the local time is within 900 seconds. A miner can profit by setting a block's timestamp to satisfy the conditions in his favor as much as possible. Check whether there are key time stamp dependent functions in the contract code implementation.

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

Safety advice: None.

3.29. Unsafe interface use [passed]

Check if unsafe interfaces are used in the contract code implementation.

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

Safety advice: None.

4. Appendix A: contract code

This test code source:

4.1. ConfluxStar.sol

```
pragma solidity =0.5.16;//knownsec//指定编译器版本，符合推荐做法

import "@openzeppelin/contracts/math/SafeMath.sol";//knownsec//使用了安全的数值计算函数

import "@openzeppelin/contracts/introspection/IERC1820Registry.sol";
import "@openzeppelin/contracts/token/ERC777/IERC777Recipient.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "@openzeppelin/contracts/token/ERC777/IERC777.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "./Pauseable.sol";
import './SponsorWhitelistControl.sol';
import './libraries/Math.sol';
/**
  * Interstellar migration
  * Welcome to Conflux Star ##Farm
  *
  */
contract ConfluxStar is Ownable, Pauseable, IERC777Recipient {//knownsec//ConfluxStar 继承自 Ownable, Pauseable, IERC777Recipient

    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    SponsorWhitelistControl constant public SPONSOR = SponsorWhitelistControl(address(0x0888000000000000000000000000000000000001));
    IERC1820Registry private _erc1820 = IERC1820Registry(0x866aCA87FF33a0ae05D2164B3D999A804F583222);
    // keccak256("ERC777TokensRecipient")
    bytes32 constant private TOKENS_RECIPIENT_INTERFACE_HASH =
0xb281fc8c12954d22544db45de3159a39272895b169a852b314f9cc762e44c53b;
    struct UserInfo {
        uint256 amount;
        uint256 rewardDebt;
    }
    struct PoolInfo {
        IERC20 lpToken;              // Address of LP token contract.
        uint256 allocPoint;          // How many allocation points assigned to this
pool. Moons to distribute per block.
        uint256 lastRewardTime;   //
        uint256 accTokenPerShare; //
    }
    // Moon crosschain atom mapping
    address public cMoonToken;
    uint256 public tokenPerSecond;
    uint256 public startFarmTime;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo;
    // Total allocation poitns. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when Token mining starts.
    uint256 public startBlock;
    mapping(address => uint256) public poolIndexs;
    mapping (address => bool) private _accountCheck;
    address[] private _accountList;
    event TokenTransfer(address indexed tokenAddress, address indexed from,
address to, uint value);
    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(//knownsec//构造函数

            address _cMoon,
            uint256 _tokenPerSecond,
            uint256 _startTime
        ) public {
            cMoonToken = _cMoon;
            tokenPerSecond = _tokenPerSecond; // Calculate the production rate
according to the mining situation
            startFarmTime = _startTime;
            _erc1820.setInterfaceImplementer(address(this),
TOKENS_RECIPIENT_INTERFACE_HASH, address(this));
            // register all users as sponsees
            address[] memory users = new address[](1);
            users[0] = address(0);
            SPONSOR.add_privilege(users);
        }
        function poolLength() external view returns (uint256) {
            return poolInfo.length;
        }
        function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate)
public onlyOwner {
            if (_withUpdate) {
                massUpdatePools();
            }
            require(poolIndexs[address(_lpToken)] < 1, "LpToken exists");
            uint256 lastRewardTime = block.timestamp > startFarmTime ?
block.timestamp : startFarmTime;
            totalAllocPoint = totalAllocPoint.add(_allocPoint);
            poolInfo.push(PoolInfo({
                lpToken: _lpToken,
                allocPoint: _allocPoint,
                lastRewardTime: lastRewardTime,
                accTokenPerShare: 0
            }));
            poolIndexs[address(_lpToken)] = poolInfo.length;
        }
        function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public
onlyOwner {
            if (_withUpdate) {
                massUpdatePools();
            }
            totalAllocPoint =
totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
            poolInfo[_pid].allocPoint = _allocPoint;
        }
        function setTokenPerSecond(uint256 _tokenPerSecond) public onlyOwner
{
            massUpdatePools();
            tokenPerSecond = _tokenPerSecond;
        }
        function pendingToken(uint256 _pid, address _user) external view returns
(uint256) {
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_user];
            uint256 accTokenPerShare = pool.accTokenPerShare;
            uint256 lpSupply = pool.lpToken.balanceOf(address(this));
```

```
            if (block.timestamp > pool.lastRewardTime && lpSupply != 0) {
                uint256 tokenReward = _getPoolReward(pool.lastRewardTime);
                accTokenPerShare                                            =
accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));
            }

            return
user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt);
        }
        function massUpdatePools() public whenPaused {
            uint256 length = poolInfo.length;
            for (uint256 pid = 0; pid < length; ++pid) {
                updatePool(pid);
            }
        }
        function updatePool(uint256 _pid) public whenPaused {
            PoolInfo storage pool = poolInfo[_pid];
            if (block.timestamp <= pool.lastRewardTime) {
                return;
            }
            uint256 lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardTime = block.timestamp;
                return;
            }
            uint256 tokenReward = _getPoolReward(pool.lastRewardTime);
            pool.accTokenPerShare                                           =
pool.accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));
            pool.lastRewardTime = block.timestamp;
        }
        //
        function  deposit(uint256  _pid,  uint256  _amount,  address  to)  public
whenPaused {
            if(to == address(0)){
                to = address(msg.sender);
            }
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][to];
            updatePool(_pid);
            if (user.amount > 0) {
                uint256                        pending                      =
user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardDebt);
                safeTokenTransfer(to, pending);
            }
            pool.lpToken.safeTransferFrom(to, address(this), _amount);
            user.amount = user.amount.add(_amount);
            user.rewardDebt                                                    =
user.amount.mul(pool.accTokenPerShare).div(1e12);
            // data migration
            if (!_accountCheck[to]) {
                _accountCheck[to] = true;
                _accountList.push(to);
            }
            emit Deposit(to, _pid, _amount);
        }
        function withdraw(uint256 _pid, uint256 _amount) public whenPaused {
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            require(_amount > 0, "user amount is zero");
            require(user.amount >= _amount, "withdraw: not good");
            updatePool(_pid);
            uint256                     pending                                =
user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardDebt);
            safeTokenTransfer(msg.sender, pending);
            user.amount = user.amount.sub(_amount);
            user.rewardDebt                                                    =
user.amount.mul(pool.accTokenPerShare).div(1e12);
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
            emit Withdraw(msg.sender, _pid, _amount);
        }
        function emergencyWithdraw(uint256 _pid) public whenPaused {
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            uint256 _amount = user.amount;
            require(_amount > 0, "user amount is zero");
            user.amount = 0;
            user.rewardDebt = 0;
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
            emit EmergencyWithdraw(msg.sender, _pid, _amount);
        }
        function moonBalance() external view returns(uint256) {
          return IERC20(cMoonToken).balanceOf(address(this));
        }
        function   safeTokenTransfer(address   _to,   uint256   _amount)   internal
{//knownsec//未验证_to 是否为空地址
            uint256 tokenBal = IERC20(cMoonToken).balanceOf(address(this));
            require(_amount <= tokenBal, "ConfluxStar: Balance insufficient");
            IERC20(cMoonToken).transfer(_to, _amount);
        }
        function _getPoolReward(uint256 _poolLastRewardTime) internal view
returns(uint256) {
            uint256 _timestamp = block.timestamp;
            return _timestamp.sub(_poolLastRewardTime).mul(tokenPerSecond);
        }
        // custodian deposit
        function tokensReceived(address operator, address from, address to, uint
amount,
              bytes calldata userData,
              bytes calldata operatorData) external {//knownsec//未验证 to 是否
为空地址
              emit TokenTransfer(msg.sender, from, to, amount);
        }
        //---------------- Data Migration ---------------------
        function accountTotal() public view returns (uint256) {
            return _accountList.length;
        }
        function accountList(uint256 begin) public view returns (address[100]
memory) {
            require(begin >= 0 && begin < _accountList.length, "MoonSwap:
accountList out of range");
            address[100] memory res;
            uint256 range = Math.min(_accountList.length, begin.add(100));
            for (uint256 i = begin; i < range; i++) {
                res[i-begin] = _accountList[i];
            }
            return res;
        }
    }
```

4.2. Migrator.sol

```
    pragma solidity 0.6.12;//knownsec//指定编译器版本，符合推荐做法

    import "./uniswapv2/interfaces/IUniswapV2Pair.sol";
    import "./uniswapv2/interfaces/IUniswapV2Factory.sol";
    import "./uniswapv2/interfaces/ICrosschainPair.sol";
    import "./uniswapv2/interfaces/ICrosschainFactory.sol";
    contract Migrator {
        address public master;
        ICrosschainFactory public factory;
        uint256 public notBeforeBlock;
        uint256 public desiredLiquidity = uint256(-1);
        mapping(address => bool) public originalFactories;

        constructor(//knownsec//构造函数

                address _master,
                address[] memory _oldFactories,
                ICrosschainFactory _factory,
                uint256 _notBeforeBlock
        ) public {
                master = _master;
                factory = _factory;
                notBeforeBlock = _notBeforeBlock;
                uint range = _oldFactories.length;
                require(range > 0, "Migrate: oldFactory Empty");
                for (uint i = 0; i < range; i++) {
                        originalFactories[_oldFactories[i]] = true;
                }
        }
        function migrate(IUniswapV2Pair orig) public returns (ICrosschainPair) {
                require(msg.sender == master, "not from master access");
                require(block.number >= notBeforeBlock, "too early to migrate");
                require(originalFactories[orig.factory()], "not from old factory");
                address token0 = orig.token0();
                address token1 = orig.token1();
                ICrosschainPair      pair      =      ICrosschainPair(factory.getPair(token0,
token1));
                if (pair == ICrosschainPair(address(0))) {
                        pair = ICrosschainPair(factory.createPair(token0, token1));
                }
                uint256 lp = orig.balanceOf(msg.sender);
                if (lp == 0) return pair;
                desiredLiquidity = lp;
                orig.transferFrom(msg.sender, address(orig), lp);
                orig.burn(address(pair));
                pair.mint(msg.sender);
                desiredLiquidity = uint256(-1);
                return pair;
        }
    }
```

4.3. UniswapV2Factory.sol

```
    pragma solidity >=0.5.16;//knownsec//指定编译器版本，符合推荐做法

    import './interfaces/IUniswapV2Factory.sol';
    import './UniswapV2Pair.sol';
    import './SponsorWhitelistControl.sol';
    import "./Pauseable.sol";
    contract UniswapV2Factory is IUniswapV2Factory, Pauseable
{//knownsec//UniswapV2Factory 继承自 IUniswapV2Factory, Pauseable

        address public feeTo;
        address public feeToSetter;
        address public migrator; // migratorFactory
        mapping(address => mapping(address => address)) public getPair;
        address[] public allPairs;
        SponsorWhitelistControl constant public SPONSOR =
SponsorWhitelistControl(address(0x0888000000000000000000000000000000000000
1));
        mapping(address => bool) public tokenBlacklist; // create pair blacklist
        event PairCreated(address indexed token0, address indexed token1, address
pair, uint);

        constructor(address _feeToSetter)//knownsec//构造函数

          Pauseable()
          public {
            feeToSetter = _feeToSetter;
            // register all users as sponsees
            address[] memory users = new address[](1);
            users[0] = address(0);
            SPONSOR.add_privilege(users);
        }
        function allPairsLength() external view returns (uint) {
            return allPairs.length;
        }
        function createPair(address tokenA, address tokenB) external whenPaused
returns (address pair) {
            require(tokenA != tokenB, 'UniswapV2:
IDENTICAL_ADDRESSES');
            require(!tokenBlacklist[tokenA] && !tokenBlacklist[tokenB],
"createPair: not allow token");
            (address token0, address token1) = tokenA < tokenB ? (tokenA,
tokenB) : (tokenB, tokenA);
            require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
            require(getPair[token0][token1] == address(0), 'UniswapV2:
PAIR_EXISTS'); // single check is sufficient
```

```
            bytes memory bytecode = type(UniswapV2Pair).creationCode;
            bytes32 salt = keccak256(abi.encodePacked(token0, token1));
            assembly {
                pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
            }
            IUniswapV2Pair(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 {//knownsec//未验证空地址
            require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
            feeTo = _feeTo;
        }
        function setFeeToSetter(address _feeToSetter) external {//knownsec//未验
证空地址
            require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
            feeToSetter = _feeToSetter;
        }
        function setMigrator(address _migrator) external {//knownsec//未验证空地
址
            require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
            migrator = _migrator;
        }
        function setTokenBlacklist(address tokenAddr, bool _status) external {
          require(msg.sender == feeToSetter, 'Factory: FORBIDDEN');
          tokenBlacklist[tokenAddr] = _status;
        }
    }
```

4.4. CrosschainPair.sol

```
    pragma solidity =0.6.12;//knownsec//指定编译器版本，符合推荐做法
    import './UniswapV2ERC20.sol';
    import './libraries/Math.sol';//knownsec//使用了安全的数值计算函数
```

```
import './libraries/UQ112x112.sol';
import './interfaces/IERC20.sol';
import './interfaces/ICrosschainFactory.sol';
import './interfaces/IUniswapV2Callee.sol';
import './interfaces/IWETH.sol';
interface IMigrator {
        // Return the desired amount of liquidity token that the migrator wants.
        function desiredLiquidity() external view returns (uint256);
}

contract CrosschainPair is UniswapV2ERC20 {//CrosschainPair 继承自
UniswapV2ERC20
        using SafeMath  for uint;
        using UQ112x112 for uint224;
        address public WETH;
        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
        struct PairMigration {
          uint migrateLiquidity;        // migrator total liquidity
          uint amount0;
          uint amount1;
        }
        PairMigration public pairMigration;
        uint private unlocked = 1;
        modifier lock() {
            require(unlocked == 1, 'UniswapV2: LOCKED');
            unlocked = 0;
            _;
            unlocked = 1;
        }
        function _safeTransfer(address token, address to, uint value) private {
            (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(SELECTOR, to, value));
```

```
        require(success && (data.length == 0 || abi.decode(data, (bool))),
'Moonswap: TRANSFER_FAILED');
    }
    function _safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'Moonswap: ETH_TRANSFER_FAILED');
    }
    receive() external payable {
        assert(msg.sender == WETH); // only accept ETH via fallback from
the WETH contract
    }
    function getReserves() public view returns (uint112 _reserve0, uint112
_reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }
    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address
indexed to);

    constructor() public {//knownsec//构造器

        factory = msg.sender;
        WETH = ICrosschainFactory(factory).WETH();
    }
    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'Moonswap: FORBIDDEN'); //
sufficient check
        token0 = _token0;
        token1 = _token1;
    }
    //
    function mint(address to) external lock returns (uint liquidity)
{//knownsec//未验证 to 是否为空地址

        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0;
        uint amount1 = balance1;
        address migrator = ICrosschainFactory(factory).migrator();
        require(msg.sender == migrator, "Moonswap: FORBIDDEN");
        liquidity = IMigrator(migrator).desiredLiquidity();
```

```
            require(liquidity > 0, 'Moonswap: INSUFFICIENT_LIQUIDITY_MINTED');
            _mint(to, liquidity);
            pairMigration.migrateLiquidity = pairMigration.migrateLiquidity.add(liquidity);
            pairMigration.amount0 = pairMigration.amount0.add(amount0);
            pairMigration.amount1 = pairMigration.amount1.add(amount1);
            // move asset for crosschain safe address audit the process
            address _receiveAddress = ICrosschainFactory(factory).getCfxReceiveAddr(address(this));
            require(_receiveAddress != address(0), 'Moonswap: receive is ZERO_ADDRESS');//knownsec//验证空地址

            if(token0 == WETH){
                IWETH(WETH).withdraw(amount0);
                _safeTransferETH(_receiveAddress, amount0);
            }else{
                _safeTransfer(token0, _receiveAddress, amount0);
            }
            if(token1 == WETH){
                IWETH(WETH).withdraw(amount1);
                _safeTransferETH(_receiveAddress, amount1);
            }else{
                _safeTransfer(token1, _receiveAddress, amount1);
            }
            emit Mint(msg.sender, amount0, amount1);
        }
    }
```

4.5. CrosschainFactory.sol

```
    pragma solidity =0.6.12;//knownsec//指定编译器版本，符合推荐做法

    import './interfaces/ICrosschainFactory.sol';
    import './interfaces/ICrosschainPair.sol';
    import './CrosschainPair.sol';
    contract CrosschainFactory is ICrosschainFactory {//knownsec//CrosschainFactory 继承自 ICrosschainFactory

        address public override migrator;
        address public override feeToSetter;
        address public override WETH;

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

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


        constructor() public {//knownsec//构造函数
            feeToSetter = msg.sender;

            uint chainId;
            assembly {
                chainId := chainid()
            }

            WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
            if( chainId == 4 ){ // rinkeby
                WETH = 0xc778417E063141139Fce010982780140Aa0cD5Ab;
            }
        }

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

        function createPair(address tokenA, address tokenB) external override
returns (address pair) {
            require(tokenA != tokenB, 'MoonSwap: IDENTICAL_ADDRESSES');
            (address token0, address token1) = tokenA < tokenB ? (tokenA,
tokenB) : (tokenB, tokenA);
            require(token0 != address(0), 'MoonSwap: ZERO_ADDRESS');
            require(getPair[token0][token1] == address(0), 'MoonSwap:
PAIR_EXISTS'); // single check is sufficient//knownsec//验证空地址

            bytes memory bytecode = type(CrosschainPair).creationCode;
            bytes32 salt = keccak256(abi.encodePacked(token0, token1));
            assembly {
                pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
            }

            ICrosschainPair(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 setMigrator(address _migrator) external override {//knownsec// 未
验证空地址
            require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
            migrator = _migrator;
        }

        // add safe conflux fund contract associated address
        function addCfxReceiveAddr(address token0, address token1, address
_receiveAddr) external {
            require(msg.sender == feeToSetter, 'Moonswap: FORBIDDEN');
            require(_receiveAddr != address(0), "MoonSwap: Receive Address is
zero");
            address pair = getPair[token0][token1];
            require(pair != address(0), "MoonSwap: Pair no exists");

            getCfxReceiveAddr[pair] = _receiveAddr;
        }

        function setFeeToSetter(address _feeToSetter) external {
            require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
            feeToSetter = _feeToSetter;
        }
        function setWETH(address _weth) external {
            require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
            require(_weth != address(0), "MoonSwap: weth is zero");
            WETH = _weth;
        }
    }
```

4.6. MigratorFactory.sol

```
    pragma solidity ^0.5.16;//knownsec//指定编译器版本，符合推荐做法


    import './SponsorWhitelistControl.sol';
```

```
import "@openzeppelin/contracts/utils/Address.sol";
import './MigratorPair.sol';
import "@openzeppelin/contracts/math/SafeMath.sol";//knownsec//使用了安全的数值计算函数
import "./Pauseable.sol";

// migrator get crosschain address, then listen transfer convert erc777 cToken
// register tokenReceive get the asset

contract MigratorFactory is Pauseable {//knownsec//MigratorFactory 继承自 Pauseable
    using Address for address;
    using SafeMath  for uint;

    address public operatorAddr;

    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;
    SponsorWhitelistControl constant public SPONSOR = SponsorWhitelistControl(address(0x0888000000000000000000000000000000000001));
    mapping(address => address) public cMoonLpToken;
    address public swapFactory;
    mapping(address => uint) public desiredLiquidity;
    mapping(address => uint) public getInflationPair; // Lp Decimals diff

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

    constructor(address _operatorAddr)//knownsec//构造函数
      Pauseable()
      public {
        operatorAddr = _operatorAddr;
        // register all users as sponsees
        address[] memory users = new address[](1);
        users[0] = address(0);
        SPONSOR.add_privilege(users);
    }

    function allPairsLength() external view returns (uint) {
```

```
                return allPairs.length;
            }

            function createPair(address tokenA, address tokenB) external whenPaused
returns (address pair) {
                require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
                require(tokenA != tokenB, 'MoonSwap: IDENTICAL_ADDRESSES');
                (address token0, address token1) = tokenA < tokenB ? (tokenA,
tokenB) : (tokenB, tokenA);
                require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
                require(getPair[token0][token1] == address(0), 'MoonSwap:
PAIR_EXISTS'); // single check is sufficient
                bytes memory bytecode = type(MigratorPair).creationCode;
                bytes32 salt = keccak256(abi.encodePacked(token0, token1));
                assembly {
                    pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
                }
                MigratorPair(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 setMoonLpToken(address token0, address token1, address
_moonLpToken) external {
                require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
                address pair = getPair[token0][token1];
                require(pair != address(0), "MoonSwap: pair no create");
                cMoonLpToken[pair] = _moonLpToken;
            }

            function setSwapFactory(address _swapFactory) external {
                require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
                swapFactory = _swapFactory;
            }

            function setOperatorAddr(address _operatorAddr) external {
                require(msg.sender == operatorAddr, 'MoonSwap: FORBIDDEN');
                operatorAddr = _operatorAddr;
            }
```

```
        function setDesiredLiquidity(address token0, address token1, uint
_desiredLiquidity, uint _multiplier) external {
            require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
            address pair = getPair[token0][token1];
            require(pair != address(0), "MoonSwap: pair no create");
            require(_multiplier > 0, "MoonSwap: multiplier must setting");
            desiredLiquidity[pair] = _desiredLiquidity.mul(_multiplier);
            getInflationPair[pair] = _multiplier;
        }

        function getDesiredLiquidity(address token0, address token1) public view
returns(uint){
          address pair = getPair[token0][token1];
          require(pair != address(0), "MoonSwap: pair no create");

          return desiredLiquidity[pair];
        }
    }
```

4.7. MigratorPair.sol

```
    pragma solidity ^0.5.16;//knownsec//指定编译器版本，符合推荐做法

    import './SponsorWhitelistControl.sol';
    import './libraries/Math.sol';
    import "@openzeppelin/contracts/math/SafeMath.sol";
    import "@openzeppelin/contracts/introspection/IERC1820Registry.sol";
    import "@openzeppelin/contracts/token/ERC777/IERC777Recipient.sol";
    import "@openzeppelin/contracts/utils/Address.sol";
    import "@openzeppelin/contracts/ownership/Ownable.sol";
    import "@openzeppelin/contracts/token/ERC777/IERC777.sol";
    import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
    import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";//knownsec//使
用了安全的数值计算函数

    interface IMigratorFactory {
        function operatorAddr() external view returns (address);
        function swapFactory() external view returns (address);
        function cMoonLpToken(address pair) external view returns (address);
        function getInflationPair(address pair) external view returns (uint);
        function getPause() external view returns (bool);
        function confluxStar() external view returns (address);
    }

    interface IUniswapV2Factory{
        function getPair(address tokenA, address tokenB) external view returns
(address pair);
    }

    interface IUniswapV2Pair {
      function mint(address to) external returns (uint);
      function transfer(address to, uint value) external returns (bool);
      function approve(address spender, uint value) external returns (bool);
    }

    interface IConfluxStar {
      function deposit(uint256 _pid, uint256 _amount, address to) external;
      function poolIndexs(address lpToken) external returns(uint256);
    }

    contract MigratorPair is IERC777Recipient {//knownsec//MigratorPair 继承自
IERC777Recipient
        using SafeMath    for uint;
        using Address for address;
        using SafeERC20 for IERC20;

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

        mapping (address => bool) private _accountCheck;
        address[] private _accountList;

        // operator upload user shareAmount for airdrop FC
        uint totalShareAmount;
        mapping(address => uint) userShareAmount;
        uint totalLpAmount;
        mapping(address => uint) userLpAmount;

        IERC1820Registry                private                 _erc1820                =
IERC1820Registry(0x866aCA87FF33a0ae05D2164B3D999A804F583222);
        // keccak256("ERC777TokensRecipient")
        bytes32 constant private TOKENS_RECIPIENT_INTERFACE_HASH =
0xb281fc8c12954d22544db45de3159a39272895b169a852b314f9cc762e44c53b;

        SponsorWhitelistControl        constant        public        SPONSOR        =
SponsorWhitelistControl(address(0x088800000000000000000000000000000000000
1));
        event TokenTransfer(address indexed tokenAddress, address indexed from,
address to, uint value);
        event ExchangeLpToken(address indexed swapPair, address indexed from,
uint value);
        event AddLiquidityEvent(address indexed swapPair, address indexed from,
uint balance0, uint balance1);


        constructor()//knownsec//构造函数

             public
        {
             factory = msg.sender;
             _erc1820.setInterfaceImplementer(address(this),
TOKENS_RECIPIENT_INTERFACE_HASH, address(this));

             // register all users as sponsees
             address[] memory users = new address[](1);
             users[0] = address(0);
             SPONSOR.add_privilege(users);
        }

        modifier whenPaused() {
            require(!IMigratorFactory(factory).getPause(),    "Pauseable:    MoonSwap
paused");
             _;
        }

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

        // user cMoonLpToken Exchange
        function exchangeLp() external {
```

```
        address cMoonLpToken = IMigratorFactory(factory).cMoonLpToken(address(this));
        require(cMoonLpToken != address(0), "Moonswap: cMoonLpToken is ZERO_ADDRESS");
        uint amount = IERC20(cMoonLpToken).balanceOf(msg.sender);
        require(amount > 0, "MoonSwap: no balance");
        IERC20(cMoonLpToken).safeTransferFrom(msg.sender, address(this), amount);

        _exchangeLp(msg.sender, amount);
    }

    function _exchangeLp(address from, uint amount) internal {
        address swapFactory = IMigratorFactory(factory).swapFactory();
        address confluxStar = IMigratorFactory(factory).confluxStar();
        address swapPair = IUniswapV2Factory(swapFactory).getPair(token0, token1);
        require(swapPair != address(0), "MoonSwap: no swap pair");//knownsec//验证空地址

        uint _multiplier = IMigratorFactory(factory).getInflationPair(address(this));
        require(_multiplier > 0, "Moonswap: multiplier is zero");
        amount = amount.mul(_multiplier); // diff decimals Inflation Amount


        if(confluxStar != address(0)){//knownsec//验证空地址

          IUniswapV2Pair(swapPair).approve(confluxStar, amount);
          uint256 _pIndex = IConfluxStar(confluxStar).poolIndexs(swapPair);
          if(_pIndex > 0){
              IConfluxStar(confluxStar).deposit(_pIndex.sub(1), amount, from);//knownsec//在 deposit 函数里面验证了空地址
          }else{
              IUniswapV2Pair(swapPair).transfer(from, amount);
          }

        }else{
          IUniswapV2Pair(swapPair).transfer(from, amount);
        }
        userLpAmount[from] = userLpAmount[from].add(amount);

        // data migration
        if (!_accountCheck[from]) {
            _accountCheck[from] = true;
            _accountList.push(from);
        }

        emit ExchangeLpToken(swapPair, from, amount);
    }

    // Add liquidity by Operator
    function addLiquidity() external {
        address operatorAddr = IMigratorFactory(factory).operatorAddr();
        require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
        address swapFactory = IMigratorFactory(factory).swapFactory();
        // when finish addLiquidity close the method
        require(swapFactory    !=    address(0),    "MoonSwap:    no    swap
factory");//knownsec//验证空地址


        address  swapPair  =  IUniswapV2Factory(swapFactory).getPair(token0,
token1);
        require(swapPair    !=    address(0),    "MoonSwap:    no    swap
pair");//knownsec//验证空地址


        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        require(balance0 > 0 && balance1 > 0, "MoonSwap: balance is zero!");

        IERC20(token0).transfer(swapPair, balance0);
        IERC20(token1).transfer(swapPair, balance1);

        uint liquidity = IUniswapV2Pair(swapPair).mint(address(this));

        totalLpAmount = liquidity;

        emit AddLiquidityEvent(swapPair, msg.sender, balance0, balance1);
    }

    function setTotalShareAmount(uint _shareAmount) external {
      require(_shareAmount > 0, "MoonSwap: shareAmount is zero");
      address operatorAddr = IMigratorFactory(factory).operatorAddr();
      require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
      totalShareAmount = _shareAmount;
    }
```

```
        function uploadUserShares(address[] calldata _users, uint[] calldata
_shareAmounts) external {
            address operatorAddr = IMigratorFactory(factory).operatorAddr();
            require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
            uint range = _users.length;
            require(range == _shareAmounts.length, "length is no match");

            for (uint i = 0; i < range; i++) {
              address _user = _users[i];
              uint _shareAmount = _shareAmounts[i];
              userShareAmount[_user] = _shareAmount;
            }
        }

        // upload user ethereum stake data Time dimension
        function getTotalShareAmount() external view returns(uint){
              return totalShareAmount;
        }

        // interface
        function getShareAmount(address to) external view returns (uint) {
            return userShareAmount[to];
        }

        function getTotalLpAmount() external view returns(uint){
              return totalLpAmount;
        }

        function getUserLpAmount(address to) external view returns(uint) {
            return userLpAmount[to];
        }

        // custodian deposit
        function tokensReceived(address operator, address from, address to, uint
amount,
                    bytes calldata userData,
                    bytes calldata operatorData) external {//knownsec//未验证 to 是否为
空地址

                    address                    cMoonLpToken                    =
IMigratorFactory(factory).cMoonLpToken(address(this));
                    if(cMoonLpToken == msg.sender) {
```

```
                    // Exchange Moonswap Liquidity
                    _exchangeLp(from, amount);
                }

                emit TokenTransfer(msg.sender, from, to, amount);
        }

        //---------------- Data Migration ----------------------
            function accountTotal() public view returns (uint256) {
                return _accountList.length;
            }

            function accountList(uint256 begin) public view returns (address[100]
memory) {
                require(begin >= 0 && begin < _accountList.length, "MigratorPair:
accountList out of range");
                address[100] memory res;
                uint256 range = Math.min(_accountList.length, begin.add(100));
                for (uint256 i = begin; i < range; i++) {
                    res[i-begin] = _accountList[i];
                }
                return res;
            }
    }
```

4.8. MoonFund.sol

```
    pragma solidity 0.6.12;//knownsec//指定编译器版本，符合推荐做法

    import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
    import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
    import "@openzeppelin/contracts/utils/EnumerableSet.sol";

    import "@openzeppelin/contracts/math/SafeMath.sol";//knownsec//使用了安全
的数值计算函数

    import "@openzeppelin/contracts/access/Ownable.sol";
    interface IMasterStar {
      function deposit(uint256 _pid, uint256 _amount) external;
    }
    interface IMoonFansToken {
        function mint(address _to, uint256 _amount) external;
    }
```

```
/**
  * Moon transfer conflux blockchain by MoonFund
  * Moon => cMoon process
  */
contract MoonFund is Ownable {//knownsec//MoonFund 继承自 Ownable
        using SafeMath for uint256;
        using SafeERC20 for IERC20;
        // receive user deposit Moon
        mapping(address => uint256) reserves;
        address public fansToken;
        address public masterStar;
        address public moonToken;
        address public confluxStarAddr; // Conflux star Address associated etherum
address
        uint256 stakePid; //
        event StakeEvent(address indexed user, uint256 indexed pid, uint256
amount);
        event MintcTokenEvent(address indexed user, uint256 indexed pid, address
_to, uint256 amount);
        event SaveEvent(address indexed user, address _to, uint256 amount);
        event RetrieveEvent(address indexed user, uint256 amount);
        constructor(//knownsec//构造函数
           address _fansToken,
           address _mastarStar,
           address _moonToken)
           public
        {
            fansToken = _fansToken;
            masterStar = _mastarStar;
            moonToken = _moonToken;
        }
        // before stake, create pool at MasterStar
        // when stake, this contract get FansToken mint operator
        function stake(uint256 _pid, uint256 _amount) external onlyOwner {
           IMoonFansToken(fansToken).mint(address(this), _amount);
           IERC20(fansToken).safeApprove(masterStar, _amount);
           IMasterStar(masterStar).deposit(_pid, _amount);
           stakePid = _pid;
           emit StakeEvent(msg.sender, _pid, _amount);
        }
        function setConfluxStarAddr(address _addr) external onlyOwner {
```

```
        require(_addr != address(0), "MoonFund: addr is zero address");//knownsec//验证空地址
        confluxStarAddr = _addr;
    }
    // the step deposit 0 masterStar,
    function mintcToken() external {
        uint256 _pid = stakePid;
        address _to = confluxStarAddr;
        require(_to != address(0), "MoonFund: addr is zero address");//knownsec//验证空地址
        uint256 beforeBalance = IERC20(moonToken).balanceOf(address(this));
        IMasterStar(masterStar).deposit(_pid, 0);
        uint256 _diffAmount = IERC20(moonToken).balanceOf(address(this)).sub(beforeBalance);
        IERC20(moonToken).safeTransfer(address(_to), _diffAmount);
        emit MintcTokenEvent(msg.sender, _pid, _to, _diffAmount);
    }
    function onlyHarvest() external onlyOwner {
        uint256 _pid = stakePid;
        IMasterStar(masterStar).deposit(_pid, 0);
    }
    function save(uint256 _amount) external {
        address _to = confluxStarAddr;
        require(_amount > 0, "MoonFund: amount is zero");
        require(_to != address(0), "MoonFund: addr is zero address");//knownsec//验证空地址
        IERC20(moonToken).safeTransferFrom(address(msg.sender), address(this), _amount);
        IERC20(moonToken).safeTransfer(address(_to), _amount);
        reserves[msg.sender] = reserves[msg.sender].add(_amount);
        emit SaveEvent(msg.sender, _to, _amount);
    }
    function retrieve(uint256 _amount) external {
        require(_amount > 0, "MoonFund: amount is zero");
        uint256 balance = IERC20(moonToken).balanceOf(address(this));
        require(_amount < balance, "MoonFund: Balance insufficient");
        uint256 _userAmount = reserves[msg.sender];
        require(_amount <= _userAmount, "MoonFund: retrieve amount overflow");
        reserves[msg.sender] = reserves[msg.sender].sub(_amount);
```

```
            IERC20(moonToken).safeTransfer(address(msg.sender), _amount);
            emit RetrieveEvent(msg.sender, _amount);
        }
        function balanceOf() external view returns(uint256){
          return IERC20(moonToken).balanceOf(address(this));
        }
    }
```

5. Appendix B: vulnerability risk rating criteria

Smart contract vulnerability rating criteria	
Vulnerability rating	Vulnerability rating description
High risk vulnerability	Loopholes that can directly cause token contract or user's capital loss, such as value overflow vulnerability that can cause token value to return to zero, false recharge vulnerability that can cause token loss by exchange, ETH or token re-entry vulnerability that can cause contract account loss, and vulnerability that can cause token contract ownership loss, such as access control defects of key functions and call Injection results in bypass of key function access control, and loopholes that can cause token contract to fail to work properly, such as denial of service vulnerability caused by sending eth to malicious address and denial of service vulnerability caused by gas exhaustion.
Medium risk loopholes	High risk vulnerabilities can be triggered by specific address, such as the value overflow vulnerability triggered by token contract owner, access control defects of non key functions, logic design defects that can not cause direct capital loss, etc.
Low risk vulnerability	The vulnerability that is difficult to trigger, the vulnerability with limited harm after triggering, such as the value overflow vulnerability that requires a large number of eth or token to trigger, the vulnerability that the attacker can't make direct profit after triggering the value overflow, and the risk of transaction sequence dependence triggered by specifying high gas.

6. Appendix C: introduction to vulnerability testing tools

6.1. Manticore

Manticore is a symbol execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum virtual machine (EVM), an EVM disassembler / assembler, and a convenient interface for automatically compiling and analyzing solidness. It also integrates etherplay, a bit of traits of bits visual disassembler for EVM bytecode, which is used for visual analysis. Like binaries, Manticore provides a simple command-line interface and a python API for analyzing EVM bytecode.

6.2. Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrance, transaction sorting dependency, and so on. More conveniently, oyente's design is modular, so it allows advanced users to implement and insert their own detection logic to check for custom properties in their contracts.

6.3. securify.sh

Securify can verify the common security problems of Ethereum smart contract, such as transaction disorder and lack of input validation. It analyzes all possible execution paths of the program while fully automated. In addition, securify also has a specific language for specifying vulnerabilities, which enables securify to pay attention to current security and other reliability issues at any time.

6.4. Echidna

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

6.5. MAIAN

Maian is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian takes a reasonable byte code and tries to establish a series of transactions to find and confirm errors.

6.6. ethersplay

Etherplay is an EVM anti assembler, which contains relevant analysis tools.

6.7. ida-evm

Ida EVM is an IDA processor module for Ethereum virtual machine (EVM).

6.8. Remix-ide

Remix is a browser based compiler and IDE that allows users to build Ethereum contracts and debug transactions using the solidity language.

6.9. Knownsec penetration tester special tool kit

Knownsec penetration tester special tool kit, developed and collected by Knownsec penetration testing engineers, includes batch automatic testing tools dedicated to testers, self-developed tools, scripts, or utility tools.

KNOWNSEC

Beijing Knownsec Information Tech. CO., LTD.

Telephone number	+86(10)400 060 9587
E-mail	sec@knownsec.com
Official website	www.knowsec.com
Address	2509, block T2-B, Wangjing SOHO, Chaoyang District, Beijing