KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester： *Knownsec blockchain security team*

Version Summary

Content	Date	Version
Editing Document	2021/08/27	V1.0

Report Information

Title	Version	Document Number	Type
StorX Smart Contract Audit Report	V1.0	fc7b1ea353c949a59d057d749a47d440	Open to project team

Copyright Notice

Table of Contents

1. Introduction

The effective test time of this report is from From **August 24**, **2021** to **August 27**, **2021** . During this period, the security and standardization of **the token code and the exchange mining pool code of the StorX smart contract** will be audited and used as the statistical basis for the report.

The scope of this smart contract security audit does not include external contract calls, new attack methods that may appear in the future, and code after contract upgrades or tampering. (With the development of the project, the smart contract may add a new pool , New functional modules, new external contract calls, etc.), does not include front-end security and server security.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3). **The smart contract code of the StorX** is comprehensively assessed as **SAFE**.

Results of this smart contract security audit: SAFE

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

Report information of this audit:

Report Number: fc7b1ea353c949a59d057d749a47d440

Report query address link:

https://attest.im/attestation/searchResult?qurey=fc7b1ea353c949a59d057d749a47d440

Target information of the StorX audit:

Target information	
Project name	StorX

Contract address	xdc5d5f074837f5d4618b3916ba74de1bf9662a3fed
Code type	Token code, defi protocol code, XDC smart contract code
Code language	Solidity

Contract documents and hash:

Contract documents	MD5
Operatable.sol	8465092e723802399f97fad46f6c7e0b
Ownable.sol	1caee14a2798e899dabada0a599b19ae
ReputationFeed.sol	6a653efdfb6425f76ae4540dfd2d5a5a
Proxy.sol	a7d91762e28770af2bb434c427cf40b5
StorXTokenBad.sol	a7d91762e28770af2bb434c427cf40b5
IERC20.sol	d787d4b670d67ce2b6845e59e6cde15f
IREF.sol	8ec9a508643528c75d7f3a11ee064cfd
Ownable.sol	e041ce163f1ee6b279250f70c85bc47d
SafeMath.sol	8debd5737bdc1edab84e470d40d5e115
StorXStaking.sol	18552db6f4c325ed7e5dfc70082e29be
BurnableToken.sol	8fc2024e6dd45153a7a2b67dc6db8dc4
Initializable.sol	462cce618ef43bbe357645c7e6a76f80
Operator.sol	1f28af51935e39c7d456aa86e0bdb9e5
Ownable.sol	1c1f908c233135dba40725d4f8669e0f
StandardToken.sol	6dfc220b60526a58ba2908deafb731f7

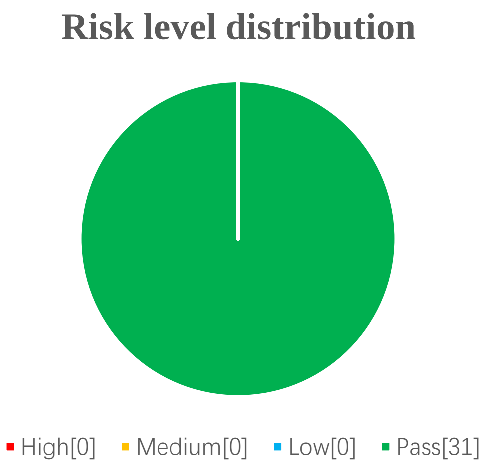
StorXToken.sol	2ce888e65df18032f7e8410e1b510da9
AddressUtils.sol	6e6278126389230a1f17955934672bf4
Migrations.sol	2e87d91a07b61383ecdff3bad31ffc53

2. Code vulnerability analysis

2.1 Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	31



2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	**Contract upgrade function**	Pass	After testing, there is no such safety vulnerability.
	Token destruction function	Pass	After testing, there is no such safety vulnerability.
	Create and delete Staker function	Pass	After testing, there is no such safety vulnerability.
	Revenue-related functions	Pass	After testing, there is no such safety vulnerability.
Basic code vulnerability detection	**Compiler version security**	Pass	After testing, there is no such safety vulnerability.
	Redundant code	Pass	After testing, there is no such safety vulnerability.
	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.
	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.oriigin authentication	Pass	After testing, there is no such safety vulnerability.

	Owner permission control	Pass	After testing, there is no such safety vulnerability.
	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.
	Low-level function safety	Pass	After testing, there is no such safety vulnerability.
	Vulnerability of additional token issuance	Pass	After testing, there is no such safety vulnerability.
	Access control defect detection	Pass	After testing, there is no such safety vulnerability.
	Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.
	Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.
	Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
	Unsafe interface use	Pass	After testing, there is no such safety vulnerability.
	Variable coverage	Pass	After testing, there is no such safety vulnerability.
	Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
	Return value call verification	Pass	After testing, there is no such safety vulnerability.

	Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.
	Timestamp dependent attack	Pass	After testing, there is no such safety vulnerability.
	Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.
	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

3. Analysis of code audit results

3.1. Contract upgrade function 【PASS】

Audit analysis: This project introduces the **proxy** mode of **openZeppelin** and uses an unstructured storage method, so that the source code can be updated after the contract is deployed, and the function of contract upgrade is realized. After auditing, the authority control is correct and the logic design is reasonable.

```
contract Proxy {
        /**
          * @dev Fallback function.
          * Implemented entirely in `_fallback`.
          */
        function() external payable {
                _fallback();
        }

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

        /**
          * @dev Delegates execution to an implementation contract.
          * This is a low level function that doesn't return to its internal call site.
          * It will return to the external caller whatever the implementation returns.
          * @param implementation Address to delegate.
          */
        function _delegate(address implementation) internal {
                assembly {
                        // Copy msg.data. We take full control of memory in this inline assembly
                        // block because it will not return to Solidity code. We overwrite the
```

```
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize)

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas, implementation, 0, calldatasize, 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize)

            switch result
                // delegatecall returns 0 on error.
                case 0 {
                    revert(0, returndatasize)
                }
                default {
                    return(0, returndatasize)
                }
        }
    }

    /**
     * @dev Function that is run as the first thing in the fallback function.
     * Can be redefined in derived contracts to add functionality.
     * Redefinitions must call super._willFallback().
     */
    function _willFallback() internal {}

    /**
     * @dev fallback implementation.
     * Extracted to enable manual triggering.
     */
    function _fallback() internal {
        _willFallback();
        _delegate(_implementation());
    }
}
```

Recommendation: nothing.

3.2. Token Burning Function 【PASS】

Audit analysis: The contract uses **_burn** to realize the function of token destruction. Its main logic first determines whether the destroyed token is less than the balance, then destroys its token, and uses the **safemath** library to perform anti-overflow verification. After auditing, the authority control is correct and the logic design is reasonable.

```
contract BurnableToken is StandardToken {
        event Burn(address indexed burner, uint256 value);

        /**
         * @dev Burns a specific amount of tokens.
         * @param _value The amount of token to be burned.
         */
        function burn(uint256 _value) public {
            _burn(msg.sender, _value);
        }

        function _burn(address _who, uint256 _value) internal {// knownsec Destroy tokens
            require(_value <= balances[_who]);
            // no need to require value <= totalSupply, since that would imply the
            // sender's balance is greater than the totalSupply, which *should* be an assertion
failure

            balances[_who] = balances[_who].sub(_value);
            totalSupply_ = totalSupply_.sub(_value);
            emit Burn(_who, _value);
            emit Transfer(_who, address(0), _value);
        }
    }
```

Recommendation: nothing.

3.3. Create and delete Staker function【PASS】

Audit analysis: The contract uses **stake** and **unstake** to realize the function of creating and deleting **Staker**. When creating a **Staker**, a certain amount of tokens are pledged and generate income. When deleting, the income is calculated and then the **Staker** is deleted. The logical judgment is rigorous. After auditing, the authority control is correct and the function design is reasonable.

```
function stake(uint256 amount_) public whenNotStaked whenNotUnStaked {// knownsec Create staker
        require(amount_ >= minStakeAmount, 'StorX: invalid amount');
        require(amount_ <= maxStakeAmount, 'StorX: invalid amount');
        require(iRepF.isStaker(msg.sender), 'StorX: sender not staker');

        stakes[msg.sender].staked = true;
        if (stakes[msg.sender].exists == false) {
            stakes[msg.sender].exists = true;
            stakes[msg.sender].stakerHolder = msg.sender;
        }

        stakeHolders.push(msg.sender);
        stakes[msg.sender].stakedTime = block.timestamp;
        stakes[msg.sender].totalRedeemed = 0;
        stakes[msg.sender].lastRedeemedAt = block.timestamp;
        stakes[msg.sender].stakedAmount = amount_;
        stakes[msg.sender].balance = 0;
        totalStaked = totalStaked.add(amount_);

        token.safeTransferFrom(msg.sender, address(this), amount_);

        emit Staked(msg.sender, amount_);
    }

    function unstake() public whenStaked whenNotUnStaked {// knownsec Remove staker
        uint256 leftoverBalance = _earned(msg.sender);
        stakes[msg.sender].unstakedTime = block.timestamp;
        stakes[msg.sender].staked = false;
        stakes[msg.sender].balance = leftoverBalance;
        stakes[msg.sender].unstaked = true;

        totalStaked = totalStaked.sub(stakes[msg.sender].stakedAmount);
        (bool exists, uint256 stakerIndex) = getStakerIndex(msg.sender);
        require(exists, 'StorX: staker does not exist');
        stakeHolders[stakerIndex] = stakeHolders[stakeHolders.length - 1];
        delete stakeHolders[stakeHolders.length - 1];
        stakeHolders.length--;

        emit Unstaked(msg.sender, stakes[msg.sender].stakedAmount);
    }
```

Recommendation: nothing.

3.4. Income-related functions【PASS】

Audit analysis: The contract uses **_earned** to calculate the income of **staked**. The pledge time is mainly calculated by the timestamp, and the pledge income is calculated using a rigorous financial model. Use **claimEarned** to calculate and obtain the actual income; use **withdrawStake** to take out the pledge and obtain the actual income. After auditing, the authority control is correct and the logic design is reasonable.

```
function _earned(address beneficiary_) internal view returns (uint256 earned) {// knownsec Income calculation
        if (stakes[beneficiary_].staked == false) return 0;
        uint256 tenure = (block.timestamp - stakes[beneficiary_].lastRedeemedAt);
        uint256 earnedStake =
            tenure
                .div(ONE_DAY)
                .mul(stakes[beneficiary_].stakedAmount)
                .mul(interest.div(interestPrecision))
                .div(100)
                .div(365);
        uint256 earnedHost = tenure.div(ONE_DAY).mul(hostingCompensation).div(365);
        earned = earnedStake.add(earnedHost);
    }

    function earned(address staker) public view returns (uint256 earnings) {
        earnings = _earned(staker);
    }

    function claimEarned(address claimAddress) public canRedeemDrip(claimAddress) {// knownsec Actual income
        require(stakes[claimAddress].staked == true, 'StorX: not staked');

        uint256 claimerReputation = iRepF.getReputation(claimAddress);
        if (claimerReputation < reputationThreshold) {
            // mark as redeemed and exit early
            stakes[claimAddress].lastRedeemedAt = block.timestamp;
            emit MissedRewards(claimAddress, reputationThreshold, claimerReputation);
            return;
        }
        // update the redeemdate even if earnings are 0
        uint256 earnings = _earned(claimAddress);

        if (earnings >= maxEarningsCap) {
            emit MaxEarningsCapReached(claimAddress, earnings, maxEarningsCap);
            earnings = maxEarningsCap;
        }

        if (earnings > 0) {
            token.mint(claimAddress, earnings);
        }

        stakes[claimAddress].totalRedeemed += earnings;
        stakes[claimAddress].lastRedeemedAt = block.timestamp;

        totalRedeemed += earnings;

        emit ClaimedRewards(claimAddress, earnings);
    }

    function withdrawStake() public whenUnStaked {//
        require(canWithdrawStake(msg.sender), 'StorX: cannot withdraw yet');
        uint256 withdrawAmount = stakes[msg.sender].stakedAmount;
        uint256 leftoverBalance = stakes[msg.sender].balance;
        token.transfer(msg.sender, withdrawAmount);
        if (leftoverBalance > 0) token.mint(msg.sender, leftoverBalance);
        stakes[msg.sender].stakedAmount = 0;
        stakes[msg.sender].balance = 0;
        stakes[msg.sender].unstaked = false;
        stakes[msg.sender].totalRedeemed += leftoverBalance;
        stakes[msg.sender].lastRedeemedAt = block.timestamp;
        totalRedeemed += leftoverBalance;
        emit WithdrewStake(msg.sender, withdrawAmount, leftoverBalance);
    }
```

Recommendation： nothing.

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

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

Audit result: After testing, the compiler version is set to 0.4.24 in the smart contract code, and there is no such security problem.

Recommendation: nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

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

Recommendation: nothing.

4.3. Use of safe arithmetic library【PASS】

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

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

Recommendation: nothing.

4.4. Not recommended encoding【PASS】

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

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

Recommendation: nothing.

4.5. Reasonable use of require/assert【PASS】

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

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

Recommendation: nothing.

4.6. Fallback function safety【PASS】

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

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

Recommendation: nothing.

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.

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

Recommendation: nothing.

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.

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

Recommendation: nothing.

4.9. Gas consumption detection【PASS】

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

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

Recommendation: nothing.

4.10. call injection attack【PASS】

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

Audit result: After testing, the smart contract does not have this vulnerability.

Recommendation: nothing.

4.11. Low-level function safety【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.

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

Recommendation: nothing.

4.12. Vulnerability of additional token issuance【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.

Audit result: After testing, the smart contract code has the function of issuing additional tokens, but it is only internally adjusted, and there is no external call method, so it is approved.

Recommendation： nothing.

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.

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

Recommendation： nothing.

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.

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

Recommendation: nothing.

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 of 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.

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

Recommendation: nothing.

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 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. 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.

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

Recommendation: nothing.

4.17. Unsafe interface usage【PASS】

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

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

Recommendation: nothing.

4.18. Variable coverage【PASS】

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

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

Recommendation: nothing.

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.

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

Recommendation: nothing.

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.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send XDC 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 Limit by passing in gas_value parameters), which 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 XDC sending failure.

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

Recommendation: nothing.

4.21. Transaction order dependency【PASS】

Since miners always get 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.

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

Recommendation: nothing.

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 only needs 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 depend on the timestamp in the contract code implementation.

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

Recommendation: nothing.

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 the smart contract, including malicious behavior as the transaction recipient, 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.

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

Recommendation: nothing.

4.24. Fake recharge vulnerability【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, enter the else logic part and return 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 sensitive function scenarios such as transfer.

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

Recommendation: nothing.

4.25. Reentry attack detection【PASS】

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

Audit results: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

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.

Audit results: After testing, the smart contract does not have this vulnerability.

Recommendation: nothing.

4.27. Rearrangement attack detection【PASS】

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, so that the attacker has the opportunity to store their own information in the contract. in.

Audit results: After testing, there are no related vulnerabilities in the smart contract code.

Recommendation: nothing.

5. Appendix A: Contract Code

Source of code for this test：

```
Operatable.sol.sol
pragma solidity 0.4.24;

import './Ownable.sol';

contract Operatable is Ownable {
        address private _operator;

        event OperatorTransferred(address indexed previousOperator, address indexed newOperator);

        constructor() public {
                _operator = msg.sender;
                emit OperatorTransferred(address(0), _operator);
        }

        function operator() public view returns (address) {
                return _operator;
        }

        modifier onlyOperator() {
                require(_operator == msg.sender, 'operator: caller is not the operator');
                _;
        }

        function isOperator() public view returns (bool) {
                return msg.sender == _operator;
        }

        function transferOperator(address newOperator_) public onlyOwner {// knownsec 设置新的 Operator,仅
Owner 可调用
                _transferOperator(newOperator_);
        }

        function _transferOperator(address newOperator_) internal {
                require(newOperator_ != address(0), 'operator: zero address given for new operator');
                emit OperatorTransferred(address(0), newOperator_);
                _operator = newOperator_;
        }
}
Ownable.sol
pragma solidity 0.4.24;

/**
  * @title Ownable
  * @dev The Ownable contract has an owner address, and provides basic authorization control
  * functions, this simplifies the implementation of "user permissions".
  */
contract Ownable {
        address public owner;

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

        /**
          * @dev The Ownable constructor sets the original `owner` of the contract to the sender
          * account.
          */
        constructor() public {
                owner = msg.sender;
        }

        /**
          * @dev Throws if called by any account other than the owner.
          */
        modifier onlyOwner() {
                require(msg.sender == owner);
                _;
        }

        /**
          * @dev Allows the current owner to relinquish control of the contract.
          * @notice Renouncing to ownership will leave the contract without an owner.
          * It will not be possible to call the functions with the `onlyOwner`
          * modifier anymore.
          */
        function renounceOwnership() public onlyOwner {// knownsec 取消当前 Owner,仅 Owner 可调用
                emit OwnershipRenounced(owner);
        owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function transferOwnership(address _newOwner) public onlyOwner {//knownsec  设置新 Owner,仅 Owner 可调用
        _transferOwnership(_newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address _newOwner) internal {
        require(_newOwner != address(0));
        emit OwnershipTransferred(owner, _newOwner);
        owner = _newOwner;
    }
}
```

ReputationFeed.sol

```
pragma solidity 0.4.24;

import './Operatable.sol';

interface IReputationFeeds {
    function setReputation(address staker, uint256 reputation) external;

    function getReputation() external returns (uint256);

    function addStaker(address staker, uint256 reputation) external;

    function removeStaker(address staker) external;
}

contract ReputationFeeds is Operatable {
    mapping(address => uint256) public reputations;
    mapping(address => bool) public isStaker;
    address[] public stakers;

    event AddedStaker(address staker, uint256 reputation);
    event RemovedStaker(address staker);

    function setReputation(address staker, uint256 reputation) public onlyOperator {
        reputations[staker] = reputation;
    }

    function getReputation(address staker) public view returns (uint256) {
        return reputations[staker];
    }

    function getAllStaker() public view returns (address[]) {
        return stakers;
    }

    function addStaker(address staker, uint256 reputation) public onlyOperator {// knownsec  增加 Staker
        (bool exists, ) = getStakerIndex(staker);
        require(exists == false, 'ReputationFeeds: staker already exists');
        stakers.push(staker);
        isStaker[staker] = true;
        reputations[staker] = reputation;
        emit AddedStaker(staker, reputation);
    }

    function removeStaker(address staker) public onlyOperator {// knownsec  移除 Staker
        (bool exists, uint256 index) = getStakerIndex(staker);
        require(exists == true, 'ReputationFeeds: staker does not exists');
        stakers[index] = stakers[stakers.length - 1];
        delete stakers[stakers.length - 1];
        stakers.length--;
        isStaker[staker] = false;
        reputations[staker] = 0;
        emit RemovedStaker(staker);
    }

    function getStakerIndex(address staker) public view returns (bool, uint256) {// knownsec  判断该地址是否为Staker
        for (uint256 i = 0; i < stakers.length; i++) {
            if (stakers[i] == staker) return (true, i);
        }
        return (false, 0);
    }
}
```

Proxy.sol

```
pragma solidity ^0.4.24;

// File: zos-lib/contracts/upgradeability/Proxy.sol

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

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

    /**
     * @dev Delegates execution to an implementation contract.
     * This is a low level function that doesn't return to its internal call site.
     * It will return to the external caller whatever the implementation returns.
     * @param implementation Address to delegate.
     */
    function _delegate(address implementation) internal {
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize)

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas, implementation, 0, calldatasize, 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize)

            switch result
                // delegatecall returns 0 on error.
                case 0 {
                    revert(0, returndatasize)
                }
                default {
                    return(0, returndatasize)
                }
        }
    }

    /**
     * @dev Function that is run as the first thing in the fallback function.
     * Can be redefined in derived contracts to add functionality.
     * Redefinitions must call super._willFallback().
     */
    function _willFallback() internal {}

    /**
     * @dev fallback implementation.
     * Extracted to enable manual triggering.
     */
    function _fallback() internal {
        _willFallback();
        _delegate(_implementation());
    }
}

// File: openzeppelin-solidity/contracts/AddressUtils.sol

/**
 * Utility library of inline functions on addresses
 */
library AddressUtils {
    /**
     * Returns whether the target address is a contract
     * @dev This function will return false if invoked during the constructor of a contract,
     * as the code is not actually created until after the constructor finishes.
     * @param addr address to check
```

```
        * @return whether the target address is a contract
        */
    function isContract(address addr) internal view returns (bool) {
        uint256 size;
        // XXX Currently there is no better way to check if there is a contract in an address
        // than to check the size of the code at that address.
        // See https://ethereum.stackexchange.com/a/14016/36603
        // for more details about how this works.
        // TODO Check this again before the Serenity release, because all addresses will be
        // contracts then.
        // solium-disable-next-line security/no-inline-assembly
        assembly {
            size := extcodesize(addr)
        }
        return size > 0;
    }
}

// File: zos-lib/contracts/upgradeability/UpgradeabilityProxy.sol

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

    /**
     * @dev Storage slot with the address of the current implementation.
     * This is the keccak-256 hash of "org.zeppelinos.proxy.implementation", and is
     * validated in the constructor.
     */
    bytes32 private constant IMPLEMENTATION_SLOT =
        0x7050c9e0f4ca769c69bd3a8ef740bc37934f8e2c036e5a723fd8ee048ed3f8c3;

    /**
     * @dev Contract constructor.
     * @param _implementation Address of the initial implementation.
     */
    constructor(address _implementation) public {
        assert(IMPLEMENTATION_SLOT == keccak256('org.zeppelinos.proxy.implementation'));

        _setImplementation(_implementation);
    }

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

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

    /**
     * @dev Sets the implementation address of the proxy.
     * @param newImplementation Address of the new implementation.
     */
    function _setImplementation(address newImplementation) private {
        require(
            AddressUtils.isContract(newImplementation),
            'Cannot set a proxy implementation to a non-contract address'
        );

        bytes32 slot = IMPLEMENTATION_SLOT;

        assembly {
            sstore(slot, newImplementation)
            }
        }
}

// File: zos-lib/contracts/upgradeability/AdminUpgradeabilityProxy.sol

/**
  * @title AdminUpgradeabilityProxy
  * @dev This contract combines an upgradeability proxy with an authorization
  * mechanism for administrative tasks.
  * All external functions in this contract must be guarded by the
  * `ifAdmin` modifier. See ethereum/solidity#3864 for a Solidity
  * feature proposal that would enable this to be done automatically.
  */
contract AdminUpgradeabilityProxy is UpgradeabilityProxy {
        /**
          * @dev Emitted when the administration has been transferred.
          * @param previousAdmin Address of the previous admin.
          * @param newAdmin Address of the new admin.
          */
        event AdminChanged(address previousAdmin, address newAdmin);

        /**
          * @dev Storage slot with the admin of the contract.
          * This is the keccak-256 hash of "org.zeppelinos.proxy.admin", and is
          * validated in the constructor.
          */
        bytes32 private constant ADMIN_SLOT =
                0x10d6a54a4754c8869d6886b5f5d7fbfa5b4522237ea5c60d11bc4e7a1ff9390b;

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

        /**
          * Contract constructor.
          * It sets the `msg.sender` as the proxy administrator.
          * @param _implementation address of the initial implementation.
          */
        constructor(address _implementation) public UpgradeabilityProxy(_implementation) {
                assert(ADMIN_SLOT == keccak256('org.zeppelinos.proxy.admin'));

                _setAdmin(msg.sender);
        }

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

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

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

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

```
        _upgradeTo(newImplementation);
    }

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

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

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

        assembly {
            sstore(slot, newAdmin)
        }
    }

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

// File: contracts/StorXTokenProxy.sol

/**
 * Copyright CENTRE SECZ 2018
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is furnished to
 * do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

/**
 * @title StorXTokenProxy
 * @dev This contract proxies FiatToken calls and enables FiatToken upgrades
 */
contract StorXTokenProxy is AdminUpgradeabilityProxy {
    constructor(address _implementation) public AdminUpgradeabilityProxy(_implementation) {}
}
```

StorXTokenBad.sol

```
pragma solidity ^0.4.24;

import '../Token/Operator.sol';
import '../Token/StandardToken.sol';
```

```
contract StorxTokenBad is StandardToken, Operator {
        string public name;
        string public symbol;
        uint8 public decimals;

        function initialize(
                string _name,
                string _symbol,
                uint8 _decimals,
                uint256 _totalSupply
        ) public initializer {// knownsec  初始化代币
                name = _name;
                symbol = _symbol;
                decimals = _decimals;
                balances[msg.sender] = _totalSupply;
                totalSupply_ = _totalSupply;

                _initializeOwner();
                _initializeOperator();
        }

        /**
          * calls internal function _mint()
          */
        function mint(address to, uint256 amount) public {
                _mint(to, amount);
        }
}
```

IERC20.sol

```
pragma solidity ^0.4.24;

/**
  * @title IERC20
  * @dev Interface to ERC20 token
  */
interface IERC20 {
        /**
          * @dev Returns the amount of tokens in existence.
          */
        function totalSupply() external view returns (uint256);

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

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

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

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

        /**
          * @dev Moves `amount` tokens from `sender` to `recipient` using the
          * allowance mechanism. `amount` is then deducted from the caller's
          * allowance.
          *
```

```
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    function mint(address to, uint256 amount) external;

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

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

IREF.sol

```
pragma solidity ^0.4.24;

interface IRepF {
    function reputations(address staker) external view returns (uint256);

    function stakers(uint256 index) external view returns (address);

    function getReputation(address staker) external view returns (uint256);

    function isStaker(address staker) external view returns (bool);

    function getStakerIndex(address staker) external view returns (bool, uint256);
}
```

Ownable.sol

```
pragma solidity ^0.4.24;

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    /// @notice
    address public owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor() public {
        owner = msg.sender;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(msg.sender == owner, 'Ownable: sender not owner');
        _;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * @notice Renouncing to ownership will leave the contract without an owner.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     */
    function renounceOwnership() public onlyOwner {// knownsec 放弃 Owner 所有权
        emit OwnershipRenounced(owner);
        owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function transferOwnership(address _newOwner) public onlyOwner {
        _transferOwnership(_newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address _newOwner) internal {// knownsec  更换 Owner
        require(_newOwner != address(0));
        emit OwnershipTransferred(owner, _newOwner);
        owner = _newOwner;
    }
}
```

SafeMath.sol

```
pragma solidity ^0.4.24;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
    /**
     * @dev Multiplies two numbers, throws on overflow.
     */
    function mul(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (_a == 0) {
            return 0;
        }

        c = _a * _b;
        assert(c / _a == _b);
        return c;
    }

    /**
     * @dev Integer division of two numbers, truncating the quotient.
     */
    function div(uint256 _a, uint256 _b) internal pure returns (uint256) {
        // assert(_b > 0); // Solidity automatically throws when dividing by 0
        // uint256 c = _a / _b;
        // assert(_a == _b * c + _a % _b); // There is no case in which this doesn't hold
        return _a / _b;
    }

    /**
     * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
     */
    function sub(uint256 _a, uint256 _b) internal pure returns (uint256) {
        assert(_b <= _a);
        return _a - _b;
    }

    /**
     * @dev Adds two numbers, throws on overflow.
     */
    function add(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
        c = _a + _b;
        assert(c >= _a);
        return c;
    }
}
```

StorXStaking.sol

```
pragma solidity ^0.4.24;

import '../AddressUtils.sol';
import './SafeMath.sol';
import './Ownable.sol';
import './IREF.sol';
import './IERC20.sol';

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    function safeTransfer(
        IERC20 _token,
        address _to,
        uint256 _value
    ) internal {
        require(_token.transfer(_to, _value));
    }

    function safeTransferFrom(
        IERC20 _token,
        address _from,
        address _to,
        uint256 _value
    ) internal {
        require(_token.transferFrom(_from, _to, _value));
    }

    function safeApprove(
        IERC20 _token,
        address _spender,
        uint256 _value
    ) internal {
        require(_token.approve(_spender, _value));
    }
}

contract StorxStaking is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using AddressUtils for address;

    uint256 private ONE_DAY = 86400;
    uint256 private ONE_YEAR_TO_DAY = 365;

    struct Stake {
        address stakerHolder;
        uint256 stakedAmount;
        bool staked;
        bool exists;
        bool unstaked;
        uint256 stakedTime;
        uint256 unstakedTime;
        uint256 totalRedeemed;
        uint256 lastRedeemedAt;
        uint256 balance; // ? required
    }

    mapping(address => Stake) public stakes;
    address[] public stakeHolders;

    IERC20 public token; // initialized in constructor
    IRepF public iRepF;
    uint256 public reputationThreshold;
    uint256 public hostingCompensation = 750 * 12 * 10**18;
    uint256 public totalStaked;
    uint256 public minStakeAmount = 1000 * 10**18;
    uint256 public maxStakeAmount = 1000000 * 10**18;
    uint256 public coolOff = ONE_DAY * 7;
    uint256 public interest;
    uint256 public totalRedeemed = 0;
    uint256 public redeemInterval = 30 * ONE_DAY;
    uint256 public maxEarningsCap = 40000 * 10**18;

    uint256 public interestPrecision = 100;

    event Staked(address staker, uint256 amount);

    event Unstaked(address staker, uint256 amount);
    event WithdrewStake(address staker, uint256 principal, uint256 earnings);
    event ClaimedRewards(address staker, uint256 amount);
    event MissedRewards(address staker, uint256 threshold, uint256 reputation);
    event MaxEarningsCapReached(address staker, uint256 earnings, uint256 cap);

    // Parameter Change Events
    event MinStakeAmountChanged(uint256 prevValue, uint256 newValue);
    event MaxStakeAmountChanged(uint256 prevValue, uint256 newValue);
    event RateChanged(uint256 prevValue, uint256 newValue);
    event CoolOffChanged(uint256 prevValue, uint256 newValue);
    event RedeemIntervalChanged(uint256 prevValue, uint256 newValue);
    event ReputationFeedChanged(address prevValue, address newValue);
    event ReputationThresholdChanged(uint256 prevValue, uint256 newValue);
    event HostingCompensationChanged(uint256 prevValue, uint256 newValue);
    event MaxEarningCapChanged(uint256 prevValue, uint256 newValue);
    event InterestPrecisionChanged(uint256 prevValue, uint256 newValue);

    event WithdrewTokens(address beneficiary, uint256 amount);
    event WithdrewXdc(address beneficiary, uint256 amount);
```

```
modifier whenStaked() {
        require(stakes[msg.sender].staked == true, 'StorX: not staked');
        _;
}

modifier whenNotStaked() {
        require(stakes[msg.sender].staked == false, 'StorX: already staked');
        _;
}

modifier whenNotUnStaked() {
        require(stakes[msg.sender].unstaked == false, 'StorX: in unstake period');
        _;
}

modifier whenUnStaked() {
        require(stakes[msg.sender].unstaked == true, 'StorX: not un-staked');
        _;
}

modifier canRedeemDrip(address staker) {
        require(stakes[staker].exists, 'StorX: staker does not exist');
        require(
                stakes[staker].lastRedeemedAt + redeemInterval <= block.timestamp,
                'StorX: cannot claim drip yet'
        );
        _;
}

function canWithdrawStake(address staker) public view returns (bool) {
        require(stakes[staker].exists, 'StorX: stakeholder does not exists');
        require(stakes[staker].staked == false, 'StorX: stakeholder still has stake');
        require(stakes[staker].unstaked == true, 'StorX: not in unstake period');
        uint256 unstakeTenure = block.timestamp - stakes[staker].unstakedTime;
        return coolOff < unstakeTenure;
}

constructor(
        IERC20 token_,
        uint256 interest_,
        IRepF reputationContract_
) public {
        token = token_;
        interest = interest_;
        iRepF = reputationContract_;
}

function stake(uint256 amount_) public whenNotStaked whenNotUnStaked {// knownsec 创建 staker
        require(amount_ >= minStakeAmount, 'StorX: invalid amount');
        require(amount_ <= maxStakeAmount, 'StorX: invalid amount');
        require(iRepF.isStaker(msg.sender), 'StorX: sender not staker');

        stakes[msg.sender].staked = true;
        if (stakes[msg.sender].exists == false) {
                stakes[msg.sender].exists = true;
                stakes[msg.sender].stakerHolder = msg.sender;
        }

        stakeHolders.push(msg.sender);
        stakes[msg.sender].stakedTime = block.timestamp;
        stakes[msg.sender].totalRedeemed = 0;
        stakes[msg.sender].lastRedeemedAt = block.timestamp;
        stakes[msg.sender].stakedAmount = amount_;
        stakes[msg.sender].balance = 0;

        totalStaked = totalStaked.add(amount_);

        token.safeTransferFrom(msg.sender, address(this), amount_);

        emit Staked(msg.sender, amount_);
}

function unstake() public whenStaked whenNotUnStaked {// knownsec 删除 staker
        uint256 leftoverBalance = _earned(msg.sender);
        stakes[msg.sender].unstakedTime = block.timestamp;
        stakes[msg.sender].staked = false;
        stakes[msg.sender].balance = leftoverBalance;
        stakes[msg.sender].unstaked = true;

        totalStaked = totalStaked.sub(stakes[msg.sender].stakedAmount);
        (bool exists, uint256 stakerIndex) = getStakerIndex(msg.sender);
        require(exists, 'StorX: staker does not exist');
        stakeHolders[stakerIndex] = stakeHolders[stakeHolders.length - 1];
        delete stakeHolders[stakeHolders.length - 1];
        stakeHolders.length--;
        emit Unstaked(msg.sender, stakes[msg.sender].stakedAmount);
    }

    function _earned(address beneficiary_) internal view returns (uint256 earned) {// knownsec  收益计算
        if (stakes[beneficiary_].staked == false) return 0;
        uint256 tenure = (block.timestamp - stakes[beneficiary_].lastRedeemedAt);
        uint256 earnedStake =
            tenure
                .div(ONE_DAY)
                .mul(stakes[beneficiary_].stakedAmount)
                .mul(interest.div(interestPrecision))
                .div(100)
                .div(365);
        uint256 earnedHost = tenure.div(ONE_DAY).mul(hostingCompensation).div(365);
        earned = earnedStake.add(earnedHost);
    }

    function earned(address staker) public view returns (uint256 earnings) {
        earnings = _earned(staker);
    }

    function claimEarned(address claimAddress) public canRedeemDrip(claimAddress) {// knownsec  实际收益
        require(stakes[claimAddress].staked == true, 'StorX: not staked');

        uint256 claimerReputation = iRepF.getReputation(claimAddress);
        if (claimerReputation < reputationThreshold) {
            // mark as redeemed and exit early
            stakes[claimAddress].lastRedeemedAt = block.timestamp;
            emit MissedRewards(claimAddress, reputationThreshold, claimerReputation);
            return;
        }

        // update the redeemdate even if earnings are 0
        uint256 earnings = _earned(claimAddress);

        if (earnings >= maxEarningsCap) {
            emit MaxEarningsCapReached(claimAddress, earnings, maxEarningsCap);
            earnings = maxEarningsCap;
        }

        if (earnings > 0) {
            token.mint(claimAddress, earnings);
        }

        stakes[claimAddress].totalRedeemed += earnings;
        stakes[claimAddress].lastRedeemedAt = block.timestamp;

        totalRedeemed += earnings;

        emit ClaimedRewards(claimAddress, earnings);
    }

    function withdrawStake() public whenUnStaked {// knownsec  取回质押，并获取收益
        require(canWithdrawStake(msg.sender), 'StorX: cannot withdraw yet');
        uint256 withdrawAmount = stakes[msg.sender].stakedAmount;
        uint256 leftoverBalance = stakes[msg.sender].balance;
        token.transfer(msg.sender, withdrawAmount);
        if (leftoverBalance > 0) token.mint(msg.sender, leftoverBalance);
        stakes[msg.sender].stakedAmount = 0;
        stakes[msg.sender].balance = 0;
        stakes[msg.sender].unstaked = false;
        stakes[msg.sender].totalRedeemed += leftoverBalance;
        stakes[msg.sender].lastRedeemedAt = block.timestamp;
        totalRedeemed += leftoverBalance;
        emit WithdrewStake(msg.sender, withdrawAmount, leftoverBalance);
    }

    function nextDripAt(address claimerAddress) public view returns (uint256) {
        require(stakes[claimerAddress].staked == true, 'StorX: address has not staked');
        return stakes[claimerAddress].lastRedeemedAt + redeemInterval;
    }

    function canWithdrawStakeIn(address staker) public view returns (uint256) {
        require(stakes[staker].exists, 'StorX: stakeholder does not exists');
        require(stakes[staker].staked == false, 'StorX: stakeholder still has stake');
        uint256 unstakeTenure = block.timestamp - stakes[staker].unstakedTime;
        if (coolOff < unstakeTenure) return 0;
        return coolOff - unstakeTenure;
    }

    function thresholdMet(address staker) public view returns (bool) {
        return iRepF.getReputation(staker) >= reputationThreshold;
    }
```

```
function getAllStakeHolder() public view returns (address[]) {
        return stakeHolders;
}

function getStakerIndex(address staker) public view returns (bool, uint256) {
        for (uint256 i = 0; i < stakeHolders.length; i++) {
                if (stakeHolders[i] == staker) return (true, i);
        }
        return (false, 0);
}

/**

Owner Functionality Starts

  */

function setMinStakeAmount(uint256 minStakeAmount_) public onlyOwner {
        require(minStakeAmount_ > 0, 'StorX: minimum stake amount should be greater than 0');
        uint256 prevValue = minStakeAmount;
        minStakeAmount = minStakeAmount_;
        emit MinStakeAmountChanged(prevValue, minStakeAmount);
}

function setMaxStakeAmount(uint256 maxStakeAmount_) public onlyOwner {
        require(maxStakeAmount_ > 0, 'StorX: maximum stake amount should be greater than 0');
        uint256 prevValue = maxStakeAmount;
        maxStakeAmount = maxStakeAmount_;
        emit MaxStakeAmountChanged(prevValue, maxStakeAmount);
}

function setRate(uint256 interest_) public onlyOwner {
        uint256 prevValue = interest;
        interest = interest_;
        emit RateChanged(prevValue, interest);
}

function setCoolOff(uint256 coolOff_) public onlyOwner {
        uint256 prevValue = coolOff;
        coolOff = coolOff_;
        emit CoolOffChanged(prevValue, coolOff);
}

function setRedeemInterval(uint256 redeemInterval_) public onlyOwner {
        uint256 prevValue = redeemInterval;
        redeemInterval = redeemInterval_;
        emit RedeemIntervalChanged(prevValue, redeemInterval);
}

function setIRepF(IRepF repAddr_) public onlyOwner {
        address prevValue = address(iRepF);
        iRepF = repAddr_;
        emit ReputationFeedChanged(prevValue, address(iRepF));
}

function setReputationThreshold(uint256 threshold) public onlyOwner {
        uint256 prevValue = reputationThreshold;
        reputationThreshold = threshold;
        emit ReputationThresholdChanged(prevValue, reputationThreshold);
}

function setHostingCompensation(uint256 hostingCompensation_) public onlyOwner {
        uint256 prevValue = hostingCompensation;
        hostingCompensation = hostingCompensation_;
        emit HostingCompensationChanged(prevValue, hostingCompensation);
}

function setMaxEarningCap(uint256 maxEarningCap_) public onlyOwner {
        uint256 prevValue = maxEarningCap_;
        maxEarningsCap = maxEarningCap_;
        emit MaxEarningCapChanged(prevValue, maxEarningsCap);
}

function setInterestPrecision(uint256 interestPrecision_) public onlyOwner {
        require(interestPrecision_ > 0, 'StorX: precision cannot be 0');
        uint256 prevValue = interestPrecision;
        interestPrecision = interestPrecision_;
        emit InterestPrecisionChanged(prevValue, interestPrecision);
}

function withdrawTokens(address beneficiary_, uint256 amount_) public onlyOwner {
        require(amount_ > 0, 'StorX: token amount has to be greater than 0');
        token.safeTransfer(beneficiary_, amount_);
        emit WithdrewTokens(beneficiary_, amount_);
}
```

```
	function withdrawXdc(address beneficiary_, uint256 amount_) public onlyOwner {
		require(amount_ > 0, 'StorX: xdc amount has to be greater than 0');
		beneficiary_.transfer(amount_);
		emit WithdrewXdc(beneficiary_, amount_);
	}

	function destroy() public onlyOwner {// knownsec  自毁函数，仅 Owner 调用
		selfdestruct(owner);
	}
}
```

BurnableToken.sol

```
pragma solidity ^0.4.24;

import './StandardToken.sol';

/**
  * @title Burnable Token
  * @dev Token that can be irreversibly burned (destroyed).
  */
contract BurnableToken is StandardToken {
	event Burn(address indexed burner, uint256 value);

	/**
	  * @dev Burns a specific amount of tokens.
	  * @param _value The amount of token to be burned.
	  */
	function burn(uint256 _value) public {
		_burn(msg.sender, _value);
	}

	function _burn(address _who, uint256 _value) internal {// knownsec  销毁代币
		require(_value <= balances[_who]);
		// no need to require value <= totalSupply, since that would imply the
		// sender's balance is greater than the totalSupply, which *should* be an assertion failure

		balances[_who] = balances[_who].sub(_value);
		totalSupply_ = totalSupply_.sub(_value);
		emit Burn(_who, _value);
		emit Transfer(_who, address(0), _value);
	}
}
```

Initializable.sol

```
pragma solidity 0.4.24;

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

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

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

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

		_;
```

```
            if (isTopLevelCall) {
                initializing = false;
            }
        }

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

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

Operator.sol

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

pragma solidity 0.4.24;

import './Ownable.sol';

contract Operator is Ownable {
        address private _operator;

        event OperatorTransferred(address indexed previousOperator, address indexed newOperator);

        function _initializeOperator() internal initializer {// knownsec  设置管理员
            _operator = msg.sender;
            emit OperatorTransferred(address(0), _operator);
        }

        function operator() public view returns (address) {
            return _operator;
        }

        modifier onlyOperator() {
            require(_operator == msg.sender, 'operator: caller is not the operator');
            _;
        }

        function isOperator() public view returns (bool) {
            return msg.sender == _operator;
        }

        function transferOperator(address newOperator_) public onlyOwner {
            _transferOperator(newOperator_);
        }

        function _transferOperator(address newOperator_) internal {// knownsec  更换管理员
            require(newOperator_ != address(0), 'operator: zero address given for new operator');
            emit OperatorTransferred(address(0), newOperator_);
            _operator = newOperator_;
        }
}
```

Ownable.sol

```
pragma solidity ^0.4.24;

import './Initializable.sol';

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable is Initializable {
        address public owner;

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

        /**
         * @dev The Ownable constructor sets the original `owner` of the contract to the sender
         * account.
         */
        function _initializeOwner() internal initializer {
            owner = msg.sender;
```

```
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * @notice Renouncing to ownership will leave the contract without an owner.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     */
    function renounceOwnership() public onlyOwner {// knownsec 放弃 Owner 所有权
        emit OwnershipRenounced(owner);
        owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function transferOwnership(address _newOwner) public onlyOwner {
        _transferOwnership(_newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address _newOwner) internal {// knownsec 更换 Owner
        require(_newOwner != address(0));
        emit OwnershipTransferred(owner, _newOwner);
        owner = _newOwner;
    }
}
```

StandardToken.sol

```
// File: contracts/token/ERC20/ERC20Basic.sol

pragma solidity ^0.4.24;

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * See https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
    function totalSupply() public view returns (uint256);

    function balanceOf(address _who) public view returns (uint256);

    function transfer(address _to, uint256 _value) public returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
}

// File: contracts/math/SafeMath.sol

pragma solidity ^0.4.24;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
    /**
     * @dev Multiplies two numbers, throws on overflow.
     */
    function mul(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (_a == 0) {
            return 0;
        }

        c = _a * _b;
        assert(c / _a == _b);
        return c;
    }

    /**
```

```
     * @dev Integer division of two numbers, truncating the quotient.
     */
    function div(uint256 _a, uint256 _b) internal pure returns (uint256) {
        // assert(_b > 0); // Solidity automatically throws when dividing by 0
        // uint256 c = _a / _b;
        // assert(_a == _b * c + _a % _b); // There is no case in which this doesn't hold
        return _a / _b;
    }

    /**
     * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
     */
    function sub(uint256 _a, uint256 _b) internal pure returns (uint256) {
        assert(_b <= _a);
        return _a - _b;
    }

    /**
     * @dev Adds two numbers, throws on overflow.
     */
    function add(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
        c = _a + _b;
        assert(c >= _a);
        return c;
    }
}

// File: contracts/token/ERC20/BasicToken.sol

pragma solidity ^0.4.24;

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
    using SafeMath for uint256;

    mapping(address => uint256) internal balances;

    uint256 internal totalSupply_;

    /**
     * @dev Total number of tokens in existence
     */
    function totalSupply() public view returns (uint256) {
        return totalSupply_;
    }

    /**
     * @dev Transfer token for a specified address
     * @param _to The address to transfer to.
     * @param _value The amount to be transferred.
     */
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_value <= balances[msg.sender]);
        require(_to != address(0));

        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(msg.sender, _to, _value);
        return true;
    }// knownsec 转账

    /**
     * @dev Gets the balance of the specified address.
     * @param _owner The address to query the the balance of.
     * @return An uint256 representing the amount owned by the passed address.
     */
    function balanceOf(address _owner) public view returns (uint256) {
        return balances[_owner];
    }// knownsec 余额
}

// File: contracts/token/ERC20/ERC20.sol

pragma solidity ^0.4.24;

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
    function allowance(address _owner, address _spender) public view returns (uint256);
    function transferFrom(
        address _from,
        address _to,
        uint256 _value
    ) public returns (bool);

    function approve(address _spender, uint256 _value) public returns (bool);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: contracts/token/ERC20/StandardToken.sol

pragma solidity ^0.4.24;

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/issues/20
 * Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, BasicToken {
    event Mint(address indexed to, uint256 amount);

    mapping(address => mapping(address => uint256)) internal allowed;

    /**
     * @dev Transfer tokens from one address to another
     * @param _from address The address which you want to send tokens from
     * @param _to address The address which you want to transfer to
     * @param _value uint256 the amount of tokens to be transferred
     */
    function transferFrom(
        address _from,
        address _to,
        uint256 _value
    ) public returns (bool) {
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);
        require(_to != address(0));

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        emit Transfer(_from, _to, _value);
        return true;
    }// knownsec 代理人转账

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
     * Beware that changing an allowance with this method brings the risk that someone may use both the old
     * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
     * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     * @param _spender The address which will spend the funds.
     * @param _value The amount of tokens to be spent.
     */
    function approve(address _spender, uint256 _value) public returns (bool) {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }// knownsec 授权

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param _owner address The address which owns the funds.
     * @param _spender address The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    function allowance(address _owner, address _spender) public view returns (uint256) {
        return allowed[_owner][_spender];
    }

    /**
     * @dev Increase the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed[_spender] == 0. To increment
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * @param _spender The address which will spend the funds.
     * @param _addedValue The amount of tokens to increase the allowance by.
     */
    function increaseApproval(address _spender, uint256 _addedValue) public returns (bool) {
        allowed[msg.sender][_spender] = (allowed[msg.sender][_spender].add(_addedValue));
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }// knownsec 追加授权金额

    /**
     * @dev Decrease the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed[_spender] == 0. To decrement
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * @param _spender The address which will spend the funds.
     * @param _subtractedValue The amount of tokens to decrease the allowance by.
     */
    function decreaseApproval(address _spender, uint256 _subtractedValue) public returns (bool) {
        uint256 oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue >= oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }// knownsec 减少授权金额

    /**
     * @dev Function to mint tokens
     * @param _to The address that will receive the minted tokens.
     * @param _amount The amount of tokens to mint.
     * @return A boolean that indicates if the operation was successful.
     */
    function _mint(address _to, uint256 _amount) internal returns (bool) {
        totalSupply_ = totalSupply_.add(_amount);
        balances[_to] = balances[_to].add(_amount);
        emit Mint(_to, _amount);
        emit Transfer(address(0), _to, _amount);
        return true;
    }// knownsec 铸币
}
```

StorXToken.sol

```
pragma solidity ^0.4.24;

import './Operator.sol';
import './BurnableToken.sol';

contract StorxToken is BurnableToken, Operator {
    string public name;
    string public symbol;
    uint8 public decimals;

    function initialize(
        string _name,
        string _symbol,
        uint8 _decimals,
        uint256 _totalSupply
    ) public initializer {// knownsec 初始化代币
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        balances[msg.sender] = _totalSupply;
        totalSupply_ = _totalSupply;

        _initializeOwner();
        _initializeOperator();
    }

    /**
     * calls internal function _mint()
     */
    function mint(address to, uint256 amount) public onlyOperator {
        _mint(to, amount);
    }

    function destroy() public onlyOwner {
        selfdestruct(owner);
    }
}
```

AddressUtils.sol

```
pragma solidity ^0.4.24;


/**
 * Utility library of inline functions on addresses
 */
library AddressUtils {
    /**
      * Returns whether the target address is a contract
      * @dev This function will return false if invoked during the constructor of a contract,
      * as the code is not actually created until after the constructor finishes.
      * @param _addr address to check
      * @return whether the target address is a contract
      */
    function isContract(address _addr) internal view returns (bool) {// knownsec 验证合约地址
        uint256 size;
        // XXX Currently there is no better way to check if there is a contract in an address
        // than to check the size of the code at that address.
        // See https://ethereum.stackexchange.com/a/14016/36603
        // for more details about how this works.
        // TODO Check this again before the Serenity release, because all addresses will be
        // contracts then.
        // solium-disable-next-line security/no-inline-assembly
        assembly { size := extcodesize(_addr) }
        return size > 0;
    }

}
```

Migrations.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.8.0;

contract Migrations {
    address public owner = msg.sender;
    uint public last_completed_migration;

    modifier restricted() {
        require(
            msg.sender == owner,
            "This function is restricted to the contract's owner"
        );
        _;
    }

    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }
}
```

6. Appendix B：Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	**Level Description**
High	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 XDC or tokens. Access loopholes, etc.; 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.; Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending XDC to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.
Medium	High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions, and logical design defects that cannot cause direct capital losses, etc.
Low	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as value overflow vulnerabilities that require a large amount of XDC or tokens to trigger, vulnerabilities where attackers cannot directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk.

7. Appendix C： Introduction to auditing tools

7.1. Manticore

Manticore is a symbolic 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 automatic compilation and analysis of Solidity. It also integrates Ethersplay, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

7.2. Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

7.3. securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a specific language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

7.4. **Echidna**

Echidna is a Haskell library designed for fuzzing EVM code.

7.5. **MAIAN**

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

7.6. **ethersplay**

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

7.7. **ida-evm**

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

7.8. **Remix-ide**

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

7.9. **Knownsec Penetration Tester Special Toolkit**

Pen-Tester tools collection is created by KnownSec team. It contains plenty of Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing