KNOWNSEC

Smart Contract Audit Report

Safe State

Low Risk

Version description

Revised man	Revised content	Revised	version number	Reviewer
Yifeng Luo	Document creation and editing	**2020/9/22**	V1.0	Haojie Xu

Document information

Name	Document version number	Number	Privacy level
SMinter Smart Contract Audit Report	V1.0	【SMinter-DMSJ-20200922】	Open project team

Copyright statement

Any text descriptions, document formats, illustrations, photographs, methods, procedures, etc. appearing in this document, unless otherwise specified, are copyrighted by Beijing Known Chuangyu Information Technology Co., Ltd. and are protected by relevant property rights and copyright laws. No individual or institution may copy or cite any fragment of this document in any way without the written permission of Beijing Chuangyu Information Technology Co., Ltd.

Knownsec

目录

1. Review 1

2. Analysis of code vulnerability 2

2.1. Distribution of vulnerability Levels 2

2.2. Audit result summary 3

3. Result analysis 4

3.1. Reentrancy【Low】 4

3.2. Arithmetic Issues【Pass】 4

3.3. Access Control【Pass】 5

3.4. Unchecked Return Values For Low Level Calls【Pass】 5

3.5. Bad Randomness【Pass】 6

3.6. Transaction ordering dependence【Low】 6

3.7. Denial of service attack detection【Pass】 7

3.8. Logical design Flaw【Pass】 7

3.9. USDT Fake Deposit Issue【Pass】 7

3.10. Adding tokens【Low】 8

3.11. Freezing accounts bypassed【Pass】 8

4. Appendix A：Contract code 9

5. Appendix B: vulnerability risk rating criteria 31

6. Appendix C：Introduction of test tool 31

6.1. Manticore..31

6.2. Oyente 32

6.3. securify.sh...32

6.4. Echidna 32

6.5. MAIAN 32

6.6. ethersplay..32

6.7. ida-evm..33

6.8. Remix-ide 33

6.9. Knownsec Penetration Tester Special Toolkit 33

Knownsec

1. Review

The effective testing time of this report is from **September 19,2020** to **September 22,2020**. During this period, the Knownsec engineers audited the safety and regulatory aspects of **SMinter** smart contract code.

In this test, engineers comprehensively analyzed common vulnerabilities of smart contracts (Chapter 3) and It was discovered that reentrancy,transaction ordering dependence, adding tokens risk; so it's evaluated as **low-risk**.

The result of the safety auditing: Pass

Since the test process is carried out in a non-production environment, all the codes are the latest backups. We communicates with the relevant interface personnel, and the relevant test operations are performed under the controllable operation risk to avoid the risks during the test..

Target information for this test:

Module name	content
Token name	SMinter
Code type	Token code
Code language	solidity

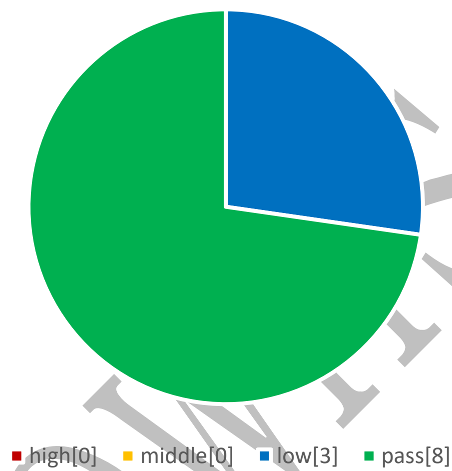
Knownsec

2. Analysis of code vulnerability

2.1. Distribution of vulnerability Levels

Vulnerability statistics			
high	Middle	low	pass
0	0	3	8

Distribution Chart



KnownSec

2.2. Audit result summary

Result			
Test project	Test content	status	description
Smart Contract	Reentrancy	Low	Check the call.value() function for security
	Arithmetic Issues	Pass	Check add and sub functions
	Access Control	Pass	Check the operation access control
	Unchecked Return Values For Low Level Calls	Pass	Check the currency conversion method.
	Bad Randomness	Pass	Check the unified content filter
	Transaction ordering dependence	Low	Check the transaction ordering dependence
	Denial of service attack detection	Pass	Check whether the code has a resource abuse problem when using a resource
	Logic design Flaw	Pass	Examine the security issues associated with business design in intelligent contract codes.
	USDT Fake Deposit Issue	Pass	Check for the existence of USDT Fake Deposit Issue
	Adding tokens	Low	It is detected whether there is a function in the token contract that may increase the total amounts of tokens
	Freezing accounts bypassed	Pass	It is detected whether there is an unverified token source account, an originating account, and whether the target account is frozen.

3. Result analysis

3.1. Reentrancy【Low】

The Reentrancy attack, probably the most famous Ethereum vulnerability，led to a hard fork of Ethereum.

When the low level call() function sends ether to the msg.sender address, it becomes vulnerable; if the address is a smart contract, the payment will trigger its fallback function with what's left of the transaction gas.

Detection results: After testing, there are related call external contract calls in the smart contract code.

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

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

Safety advice:

(1) Try to use send() and transfer() functions.

(2) If you use a low-level calling function like the call() function, you should perform the internal state change first, and then use the low-level calling function.

(3) Try to avoid calling external contracts when writing smart contracts.

3.2. Arithmetic Issues【Pass】

Also known as integer overflow and integer underflow. Solidity can handle up to 256 digits (2^256-1), The largest number increases by 1 will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum numeric value.

Integer overflows and underflows are not a new class of vulnerability, but they are especially dangerous in smart contracts. Overflow can lead to incorrect results,

Knownsec

especially if the probability is not expected, which may affect the reliability and security of the program.

Test results: No related vulnerabilities in smart contract code

Safety advice: None

3.3. Access Control【Pass】

Access Control issues are common in all programs,Also smart contracts. The famous Parity Wallet smart contract has been affected by this issue.

Test results: No related vulnerabilities in smart contract code

Safety advice: None。

3.4. Unchecked Return Values For Low Level Calls【Pass】

Also known as or related to silent failing sends, unchecked-send. There are transfer methods such as transfer(), send(), and call.value() in Solidity and can be used to send Ether to an address. The difference is: transfer will be thrown when failed to send, and rollback; only 2300gas will be passed for call to prevent reentry attacks; send will return false if send fails; only 2300gas will be passed for call to prevent reentry attacks; If .value fails to send, it will return false; passing all available gas calls (which can be restricted by passing in the gas_value parameter) cannot effectively prevent reentry attacks.

If the return value of the send and call.value switch functions is not been checked in the code, the contract will continue to execute the following code,and it may have caused unexpected results due to Ether sending failure.

Test results: No related vulnerabilities in smart contract code

Safety advice: None。

Knownsec

3.5. Bad Randomness【Pass】

Smart Contract May Need to Use Random Numbers. While Solidity offers functions and variables that can access apparently hard-to-predict values just as block.number and block.timestamp. they are generally either more public than they seem or subject to miners' influence. Because these sources of randomness are to an extent predictable, malicious users can generally replicate it and attack the function relying on its unpredictablility.

Test results: No related vulnerabilities in smart contract code

Safety advice: None。

3.6. Transaction ordering dependence【Low】

Since miners always get rewarded via gas fees for running code on behalf of externally owned addresses (EOA), users can specify higher fees to have their transactions mined more quickly. Since the Ethereum blockchain is public, everyone can see the contents of others' pending transactions.

This means if a given user is revealing the solution to a puzzle or other valuable secret, a malicious user can steal the solution and copy their transaction with higher fees to preempt the original solution.

Test results: After testing, there are related vulnerabilities in the smart contract code.

contracts\ERC20.sol lines 74

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

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

Safety advice:

Knownsec

1. When the approve function changes the quota from N to M, it can only be changed from N to 0, and then from 0 to M: require((_value == 0) || (allowance[msg.sender][_spender] == 0));

2. Use increaseApproval function and decreaseApproval function instead of approve function.

3.7. Denial of service attack detection【Pass】

In the ethernet world, denial of service is deadly, and smart contracts under attack of this type may never be able to return to normal. There may be a number of reasons for a denial of service in smart contracts, including malicious behavior as a recipient of transactions, gas depletion caused by artificially increased computing gas, and abuse of access control to access the private components of the intelligent contract. Take advantage of confusion and neglect, etc.

Test results: No related vulnerabilities in smart contract code.

Safety advice: None.

3.8. Logical design Flaw【Pass】

Detect the security problems related to business design in the contract code.

Test results: No related vulnerabilities in smart contract code.

Safety advice: None.

3.9. USDT Fake Deposit Issue【Pass】

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

Knownsec

Detection results: No related vulnerabilities in smart contract code..

Safety advice: none

3.10. Adding tokens 【Low】

It is detected whether there is a function in the token contract that may increase the total amount of tokens after the total amount of tokens is initialized.

Detection results: After testing, there are related vulnerabilities in the smart contract code.

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

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

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

Safety advice:

This issue is not a security issue, but some exchanges will restrict the use of additional issuance functions, and the specific circumstances need to be determined according to the requirements of the exchange.

3.11. Freezing accounts bypassed 【Pass】

In the token contract, when transferring the token, it is detected whether there is an unverified token source account, an originating account, and whether the target account is frozen.

Detection results: No related vulnerabilities in smart contract code.

Safety advice: none.

Knownsec

4. Appendix A：Contract code

```
•// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;
//pragma experimental ABIEncoderV2;

import "./SToken.sol";
import "./Governable.sol";

import "./TransferHelper.sol";


interface Minter {
   event Minted(address indexed recipient, address reward_contract, uint minted);

   function token() external view returns (address);
   function controller() external view returns (address);
   function minted(address, address) external view returns (uint);
   function allowed_to_mint_for(address, address) external view returns (bool);

   function mint(address gauge) external;
   function mint_many(address[8] calldata gauges) external;
   function mint_for(address gauge, address _for) external;
   function toggle_approve_mint(address minting_user) external;
}

interface LiquidityGauge {
   event Deposit(address indexed provider, uint value);
   event Withdraw(address indexed provider, uint value);
   event UpdateLiquidityLimit(address user, uint original_balance, uint
original_supply, uint working_balance, uint working_supply);

   function user_checkpoint (address addr) external returns (bool);
   function claimable_tokens(address addr) external view returns (uint);
   function claimable_reward(address addr) external view returns (uint);
   function integrate_checkpoint()      external view returns (uint);

   function kick(address addr) external;
   function set_approve_deposit(address addr, bool can_deposit) external;
   function deposit(uint _value) external;
   function deposit(uint _value, address addr) external;
   function withdraw(uint _value) external;
   function withdraw(uint _value, bool claim_rewards) external;
   function claim_rewards() external;
   function claim_rewards(address addr) external;

   function minter()                   external view returns (address);
   function crv_token()                external view returns (address);
   function lp_token()                 external view returns (address);
   function controller()               external view returns (address);
   function voting_escrow()            external view returns (address);
   function balanceOf(address)         external view returns (uint);
   function totalSupply()              external view returns (uint);
   function future_epoch_time()        external view returns (uint);
   function approved_to_deposit(address, address)   external view returns (bool);
   function working_balances(address)     external view returns (uint);
   function working_supply()           external view returns (uint);
   function period()                   external view returns (int128);
   function period_timestamp(uint)       external view returns (uint);
   function integrate_inv_supply(uint)     external view returns (uint);
   function integrate_inv_supply_of(address) external view returns (uint);
   function integrate_checkpoint_of(address) external view returns (uint);
   function integrate_fraction(address)    external view returns (uint);
   function inflation_rate()           external view returns (uint);
   function reward_contract()          external view returns (address);
   function rewarded_token()           external view returns (address);
   function reward_integral()          external view returns (uint);
   function reward_integral_for(address)   external view returns (uint);
   function rewards_for(address)         external view returns (uint);
   function claimed_rewards_for(address)   external view returns (uint);
}
```

```
    contract SSimpleGauge is LiquidityGauge, Configurable {
        using SafeMath for uint;
        using TransferHelper for address;

        address override public minter;
        address override public crv_token;
        address override public lp_token;
        address override public controller;
        address override public voting_escrow;
        mapping(address => uint) override public balanceOf;
        uint override public totalSupply;
        uint override public future_epoch_time;

        // caller -> recipient -> can deposit?
        mapping(address => mapping(address => bool)) override public
approved_to_deposit;

        mapping(address => uint) override public working_balances;
        uint override public working_supply;

        // The goal is to be able to calculate ∫(rate * balance / totalSupply dt) from
0 till checkpoint
        // All values are kept in units of being multiplied by 1e18
        int128 override public period;
        uint256[100000000000000000000000000000] override public period_timestamp;

        // 1e18 * ∫(rate(t) / totalSupply(t) dt) from 0 till checkpoint
        uint256[100000000000000000000000000000] override public integrate_inv_supply;
// bump epoch when rate() changes

        // 1e18 * ∫(rate(t) / totalSupply(t) dt) from (last_action) till checkpoint
        mapping(address => uint) override public integrate_inv_supply_of;
        mapping(address => uint) override public integrate_checkpoint_of;

        // ∫(balance * rate(t) / totalSupply(t) dt) from 0 till checkpoint
        // Units: rate * t = already number of coins per address to issue
        mapping(address => uint) override public integrate_fraction;

        uint override public inflation_rate;

        // For tracking external rewards
        address override public reward_contract;
        address override public rewarded_token;

        uint override public reward_integral;
        mapping(address => uint) override public reward_integral_for;
        mapping(address => uint) override public rewards_for;
        mapping(address => uint) override public claimed_rewards_for;


     uint public span;
     uint public end;

     function initialize(address governor, address _minter, address _lp_token) public
initializer {
         super.initialize(governor);

         minter      = _minter;
         crv_token   = Minter(_minter).token();
         lp_token    = _lp_token;
         IERC20(lp_token).totalSupply();        // just check
     }

        function setSpan(uint _span, bool isLinear) virtual external governance {
            span = _span;
            if(isLinear)
                end = now + _span;
            else
                end = 0;
        }

        function kick(address addr) virtual override external {
            _checkpoint(addr, true);
        }
```

knownsec

```
        function set_approve_deposit(address addr, bool can_deposit) virtual override
external {
            approved_to_deposit[addr][msg.sender] = can_deposit;
        }

        function deposit(uint amount) virtual override external {
            deposit(amount, msg.sender);
        }
        function deposit(uint amount, address addr) virtual override public {
            require(addr == msg.sender || approved_to_deposit[msg.sender][addr], 'Not
approved');

            _checkpoint(addr, true);

            _deposit(addr, amount);

            balanceOf[addr] = balanceOf[addr].add(amount);
            totalSupply = totalSupply.add(amount);

            emit Deposit(addr, amount);
        }
        function _deposit(address addr, uint amount) virtual internal {
            lp_token.safeTransferFrom(addr, address(this), amount);
        }

        function withdraw() virtual  external {
            withdraw(balanceOf[msg.sender], true);
        }
        function withdraw(uint amount) virtual override external {
            withdraw(amount, true);
        }
        function withdraw(uint amount, bool claim_rewards) virtual override public {
            _checkpoint(msg.sender, claim_rewards);

            totalSupply = totalSupply.sub(amount);
            balanceOf[msg.sender] = balanceOf[msg.sender].sub(amount);

            _withdraw(msg.sender, amount);

            emit Withdraw(msg.sender, amount);
        }
        function _withdraw(address to, uint amount) virtual internal {
            lp_token.safeTransfer(to, amount);
        }

        function claimable_reward(address) virtual override public view returns (uint)
{
            return 0;
        }

        function claim_rewards() virtual override public {
            return claim_rewards(msg.sender);
        }
        function claim_rewards(address) virtual override public {
            return;
        }
        function _checkpoint_rewards(address, bool) virtual internal {
            return;
        }

        function claimable_tokens(address addr) virtual override public view returns
(uint amount) {
            if(span == 0 || totalSupply == 0)
                return 0;

            amount = SMinter(minter).quotas(address(this));
            amount = amount.mul(balanceOf[addr]).div(totalSupply);

            uint lasttime = integrate_checkpoint_of[addr];
            if(end == 0) {                                                      //
isNonLinear, endless
                if(now.sub(lasttime) < span)
                    amount = amount.mul(now.sub(lasttime)).div(span);
            }else if(now < end)
                amount = amount.mul(now.sub(lasttime)).div(end.sub(lasttime));
            else if(lasttime >= end)
                amount = 0;
```

Knownsec

```
    }

    function _checkpoint(address addr, uint amount) virtual internal {
        if(amount > 0) {
            integrate_fraction[addr] = integrate_fraction[addr].add(amount);

            address teamAddr = address(config['teamAddr']);
            uint teamRatio = config['teamRatio'];
            if(teamAddr != address(0) && teamRatio != 0)
                integrate_fraction[teamAddr] =
integrate_fraction[teamAddr].add(amount.mul(teamRatio).div(1 ether));
        }
    }

    function _checkpoint(address addr, bool _claim_rewards) virtual internal {
        uint amount = claimable_tokens(addr);
        _checkpoint(addr, amount);
        _checkpoint_rewards(addr, _claim_rewards);

        integrate_checkpoint_of[addr] = now;
    }

    function user_checkpoint(address addr) virtual override external returns (bool)
{
        _checkpoint(addr, true);
        return true;
    }

    function integrate_checkpoint() override external view returns (uint) {
        return now;
    }
}

contract SExactGauge is LiquidityGauge, Configurable {
    using SafeMath for uint;
    using TransferHelper for address;

    bytes32 internal constant _devAddr_         = 'devAddr';
    bytes32 internal constant _devRatio_        = 'devRatio';
    bytes32 internal constant _ecoAddr_         = 'ecoAddr';
    bytes32 internal constant _ecoRatio_        = 'ecoRatio';

    address override public minter;
    address override public crv_token;
    address override public lp_token;
    address override public controller;
    address override public voting_escrow;
    mapping(address => uint) override public balanceOf;
    uint override public totalSupply;
    uint override public future_epoch_time;

    // caller -> recipient -> can deposit?
    mapping(address => mapping(address => bool)) override public
approved_to_deposit;

    mapping(address => uint) override public working_balances;
    uint override public working_supply;

    // The goal is to be able to calculate ∫(rate * balance / totalSupply dt) from
0 till checkpoint
    // All values are kept in units of being multiplied by 1e18
    int128 override public period;
    uint256[100000000000000000000000000000] override public period_timestamp;

    // 1e18 * ∫(rate(t) / totalSupply(t) dt) from 0 till checkpoint
    uint256[100000000000000000000000000000] override public integrate_inv_supply;
// bump epoch when rate() changes

    // 1e18 * ∫(rate(t) / totalSupply(t) dt) from (last_action) till checkpoint
    mapping(address => uint) override public integrate_inv_supply_of;
    mapping(address => uint) override public integrate_checkpoint_of;

    // ∫(balance * rate(t) / totalSupply(t) dt) from 0 till checkpoint
    // Units: rate * t = already number of coins per address to issue
    mapping(address => uint) override public integrate_fraction;
```

Knownsec

```
        uint override public inflation_rate;

        // For tracking external rewards
        address override public reward_contract;
        address override public rewarded_token;

        uint override public reward_integral;
        mapping(address => uint) override public reward_integral_for;
        mapping(address => uint) override public rewards_for;
        mapping(address => uint) override public claimed_rewards_for;

    uint public span;
    uint public end;
    mapping(address => uint) public sumMiningPerOf;
    uint public sumMiningPer;
    uint public bufReward;
    uint public lasttime;

    function initialize(address governor, address _minter, address _lp_token) public
initializer {
        super.initialize(governor);

        minter     = _minter;
        crv_token  = Minter(_minter).token();
        lp_token   = _lp_token;
        IERC20(lp_token).totalSupply();              // just check
    }

        function setSpan(uint _span, bool isLinear) virtual external governance {
           span = _span;
           if(isLinear)
              end = now + _span;
           else
              end = 0;
           lasttime = now;
        }

        function kick(address addr) virtual override external {
           _checkpoint(addr, true);
        }

        function set_approve_deposit(address addr, bool can_deposit) virtual override
external {
           approved_to_deposit[addr][msg.sender] = can_deposit;
        }

        function deposit(uint amount) virtual override external {
           deposit(amount, msg.sender);
        }
        function deposit(uint amount, address addr) virtual override public {
           require(addr == msg.sender || approved_to_deposit[msg.sender][addr], 'Not
approved');

           _checkpoint(addr, true);

           _deposit(addr, amount);

           balanceOf[msg.sender] = balanceOf[msg.sender].add(amount);
           totalSupply = totalSupply.add(amount);

           emit Deposit(msg.sender, amount);
        }
        function _deposit(address addr, uint amount) virtual internal {
           lp_token.safeTransferFrom(addr, address(this), amount);
        }

        function withdraw() virtual external {
           withdraw(balanceOf[msg.sender], true);
        }
        function withdraw(uint amount) virtual override external {
           withdraw(amount, true);
        }
        function withdraw(uint amount, bool _claim_rewards) virtual override public {
           _checkpoint(msg.sender, _claim_rewards);

           totalSupply = totalSupply.sub(amount);
           balanceOf[msg.sender] = balanceOf[msg.sender].sub(amount);
```

Knownsec

```
        _withdraw(msg.sender, amount);

        emit Withdraw(msg.sender, amount);
    }
    function _withdraw(address to, uint amount) virtual internal {
        lp_token.safeTransfer(to, amount);
    }

    function claimable_reward(address addr) virtual override public view returns
(uint) {
        addr;
        return 0;
    }

    function claim_rewards() virtual override public {
        return claim_rewards(msg.sender);
    }
    function claim_rewards(address) virtual override public {
        return;
    }
    function _checkpoint_rewards(address, bool) virtual internal {
        return;
    }

    function claimable_tokens(address addr) virtual override public view returns
(uint) {
        return _claimable_tokens(addr, claimableDelta(), sumMiningPer,
sumMiningPerOf[addr]);
    }
    function _claimable_tokens(address addr, uint delta, uint sumPer, uint
lastSumPer) virtual internal view returns (uint amount) {
        if(span == 0 || totalSupply == 0)
            return 0;

        amount = sumPer.sub(lastSumPer);
        amount = amount.add(delta.mul(1 ether).div(totalSupply));
        amount = amount.mul(balanceOf[addr]).div(1 ether);
    }
    function claimableDelta() virtual internal view returns(uint amount) {
        amount = SMinter(minter).quotas(address(this)).sub(bufReward);

        if(end == 0) {                                              //
isNonLinear, endless
            if(now.sub(lasttime) < span)
                amount = amount.mul(now.sub(lasttime)).div(span);
        }else if(now < end)
            amount = amount.mul(now.sub(lasttime)).div(end.sub(lasttime));
        else if(lasttime >= end)
            amount = 0;
    }

    function _checkpoint(address addr, uint amount) virtual internal {
        if(amount > 0) {
            integrate_fraction[addr] = integrate_fraction[addr].add(amount);

            addr = address(config[_devAddr_]);
            uint ratio = config[_devRatio_];
            if(addr != address(0) && ratio != 0)
                integrate_fraction[addr] =
integrate_fraction[addr].add(amount.mul(ratio).div(1 ether));

            addr = address(config[_ecoAddr_]);
            ratio = config[_ecoRatio_];
            if(addr != address(0) && ratio != 0)
                integrate_fraction[addr] =
integrate_fraction[addr].add(amount.mul(ratio).div(1 ether));
        }
    }

    function _checkpoint(address addr, bool _claim_rewards) virtual internal {
        if(span == 0 || totalSupply == 0)
            return;

        uint delta = claimableDelta();
        uint amount = _claimable_tokens(addr, delta, sumMiningPer,
sumMiningPerOf[addr]);
```

Knownsec

```
            if(delta != amount)
                bufReward = bufReward.add(delta).sub(amount);
            if(delta > 0)
                sumMiningPer = sumMiningPer.add(delta.mul(1 ether).div(totalSupply));
            if(sumMiningPerOf[addr] != sumMiningPer)
                sumMiningPerOf[addr] = sumMiningPer;
            lasttime = now;

            _checkpoint(addr, amount);
            _checkpoint_rewards(addr, _claim_rewards);
        }

        function user_checkpoint(address addr) virtual override external returns (bool)
{
            _checkpoint(addr, true);
            return true;
        }

        function integrate_checkpoint() override external view returns (uint) {
            return lasttime;
        }
    }


    contract SNestGauge is SExactGauge {
     address[] public rewards;
     mapping(address => mapping(address =>uint)) public sumRewardPerOf;      //
recipient => rewarded_token => can sumRewardPerOf
     mapping(address => uint) public sumRewardPer;                          //
rewarded_token => can sumRewardPerOf

     function initialize(address governor, address _minter, address _lp_token, address
_nestGauge, address[] memory _moreRewards) public initializer {
        super.initialize(governor, _minter, _lp_token);

        reward_contract = _nestGauge;
        rewarded_token  = LiquidityGauge(_nestGauge).crv_token();
        rewards         = _moreRewards;
        rewards.push(rewarded_token);
        address rewarded_token2 = LiquidityGauge(_nestGauge).rewarded_token();
        if(rewarded_token2 != address(0))
            rewards.push(rewarded_token2);

        LiquidityGauge(_nestGauge).integrate_checkpoint();      // just check
        for(uint i=0; i<_moreRewards.length; i++)
           IERC20(_moreRewards[i]).totalSupply();             // just check
     }

        function _deposit(address from, uint amount) virtual override internal {
            super._deposit(from, amount);                          //
lp_token.safeTransferFrom(from, address(this), amount);
            lp_token.safeApprove(reward_contract, amount);
            LiquidityGauge(reward_contract).deposit(amount);
        }

        function _withdraw(address to, uint amount) virtual override internal {
            LiquidityGauge(reward_contract).withdraw(amount);
            super._withdraw(to, amount);                           //
lp_token.safeTransfer(to, amount);
        }

        function claim_rewards(address to) virtual override public {
            if(span == 0 || totalSupply == 0)
                return;

            uint[] memory bals = new uint[](rewards.length);
            for(uint i=0; i<bals.length; i++)
                bals[i] = IERC20(rewards[i]).balanceOf(address(this));

            Minter(LiquidityGauge(reward_contract).minter()).mint(reward_contract);
            LiquidityGauge(reward_contract).claim_rewards();

            for(uint i=0; i<bals.length; i++) {
                uint delta = IERC20(rewards[i]).balanceOf(address(this)).sub(bals[i]);
                uint amount = _claimable_tokens(msg.sender, delta,
sumRewardPer[rewards[i]], sumRewardPerOf[msg.sender][rewards[i]]);
```

```
            if(delta > 0)
                sumRewardPer[rewards[i]] = sumRewardPer[rewards[i]].add(delta.mul(1
ether).div(totalSupply));
            if(sumRewardPerOf[msg.sender][rewards[i]] != sumRewardPer[rewards[i]])
                sumRewardPerOf[msg.sender][rewards[i]] = sumRewardPer[rewards[i]];

            if(amount > 0) {
                rewards[i].safeTransfer(to, amount);
                if(rewards[i] == rewarded_token) {
                    rewards_for[to] = rewards_for[to].add(amount);
                    claimed_rewards_for[to] = claimed_rewards_for[to].add(amount);
                }
            }
        }
    }

    function claimable_reward(address addr) virtual override public view returns
(uint) {
        uint delta =
LiquidityGauge(reward_contract).claimable_tokens(address(this));
        return _claimable_tokens(addr, delta, sumRewardPer[rewarded_token],
sumRewardPerOf[addr][rewarded_token]);
    }

    function claimable_reward2(address addr) virtual public view returns (uint) {
        uint delta =
LiquidityGauge(reward_contract).claimable_reward(address(this));
        address reward2 = LiquidityGauge(reward_contract).rewarded_token();
        return _claimable_tokens(addr, delta, sumRewardPer[reward2],
sumRewardPerOf[addr][reward2]);
    }

}


contract SMinter is Minter, Configurable {
    using SafeMath for uint;
    using Address for address payable;
    using TransferHelper for address;

   bytes32 internal constant _allowContract_   = 'allowContract';
   bytes32 internal constant _allowlist_       = 'allowlist';
   bytes32 internal constant _blocklist_       = 'blocklist';

    address override public token;
    address override public controller;
    mapping(address => mapping(address => uint)) override public minted;
// user => reward_contract => value
    mapping(address => mapping(address => bool)) override public
allowed_to_mint_for;      // minter => user => can mint?
    mapping(address => uint) public quotas;
// reward_contract => quota;

    function initialize(address governor, address token_) public initializer {
        super.initialize(governor);
        token = token_;
    }

    function setGaugeQuota(address gauge, uint quota) public governance {
       quotas[gauge] = quota;
    }

    function mint(address gauge) virtual override public {
        mint_for(gauge, msg.sender);
    }

    function mint_many(address[8] calldata gauges) virtual override external {
        for(uint i=0; i<gauges.length; i++)
            mint(gauges[i]);
    }

    function mint_for(address gauge, address _for) virtual override public {
        require(_for == msg.sender || allowed_to_mint_for[msg.sender][_for], 'Not
approved');
        require(quotas[gauge] > 0, 'No quota');
```

Knownsec

```
        require(getConfig(_blocklist_, msg.sender) == 0, 'In blocklist');
        bool isContract = msg.sender.isContract();
        require(!isContract || config[_allowContract_] != 0 ||
getConfig(_allowlist_, msg.sender) != 0, 'No allowContract');

        LiquidityGauge(gauge).user_checkpoint(_for);
        uint total_mint = LiquidityGauge(gauge).integrate_fraction(_for);
        uint to_mint = total_mint.sub(minted[_for][gauge]);

        if(to_mint != 0) {
            quotas[gauge] = quotas[gauge].sub(to_mint);
            token.safeTransfer(_for, to_mint);
            minted[_for][gauge] = total_mint;

            emit Minted(_for, gauge, total_mint);
        }
    }

    function toggle_approve_mint(address minting_user) virtual override external {
        allowed_to_mint_for[minting_user][msg.sender]
= !allowed_to_mint_for[minting_user][msg.sender];
    }
}

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

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

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

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

pragma solidity ^0.6.0;

/**
 * @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 Modifier to use in the initializer function of a contract when upgrade
EVEN times.
     */
    modifier initializerEven() {
      require(initializing || isConstructor() || initialized, "Contract instance has
already been initialized EVEN times");

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

      _;

      if (isTopLevelCall) {
        initializing = false;
      }
    }

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

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


  contract Governable is Initializable {
      address public governor;

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

      /**
       * @dev Contract initializer.
```

Knownsec

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

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

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

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

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


    contract Configurable is Governable {

        mapping (bytes32 => uint) internal config;

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

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

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


Knownsec

```
        _setConfig(bytes32(uint(key) ^ uint(addr)), value);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

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

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


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

        return c;
    }

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

    /**
```

Knownsec

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

        return c;
    }

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

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

        return c;
    }

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

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with
custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
```

Knownsec

```
        // assert(a == b * c + a % b); // There is no case in which this doesn't
hold

        return c;
    }

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

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

   function sqrt(uint x)public pure returns(uint y) {
        uint z = (x + 1) / 2;
        y = x;
        while (z < y) {
            y = z;
            z = (x / z + z) / 2;
        }
    }
}


/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
```

Knownsec

```
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

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

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-
variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns
(bytes memory) {
       return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
```

knownsec

```
         * - the calling contract must have an ETH balance of at least `value`.
         * - the called Solidity function must be `payable`.
         *
         * _Available since v3.1._
         */
        function functionCallWithValue(address target, bytes memory data, uint256 
value) internal returns (bytes memory) {
            return functionCallWithValue(target, data, value, "Address: low-level call 
with value failed");
        }

        /**
         * @dev Same as {xref-Address-functionCallWithValue-address-bytes-
uint256-}[`functionCallWithValue`], but
         * with `errorMessage` as a fallback revert reason when `target` reverts.
         *
         * _Available since v3.1._
         */
        function functionCallWithValue(address target, bytes memory data, uint256 
value, string memory errorMessage) internal returns (bytes memory) {
            require(address(this).balance >= value, "Address: insufficient balance for 
call");
            return _functionCallWithValue(target, data, value, errorMessage);
        }

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

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

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


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

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

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

        /**
```

Knownsec

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

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

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

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

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


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

Knownsec

```
     * allowances. See {IERC20-approve}.
     */
    contract ERC20 is Context, IERC20 {
        using SafeMath for uint256;
        using Address for address;

        mapping (address => uint256) public _balances;

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

        uint256 public _totalSupply;

        string internal _name;
        string internal _symbol;
        uint8 internal _decimals;

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

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

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

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

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

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

Knownsec

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

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

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

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

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

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
```

Knownsec

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

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

        _beforeTokenTransfer(sender, recipient, amount);

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

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

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

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

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
```

Knownsec

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

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

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

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

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

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

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

}


contract SfgToken is ERC20 {

 constructor(address SfgFarm) ERC20("Stable Finance Governance Token", "SFG")
public {
     uint8 decimals = 18;
     _setupDecimals(decimals);

     _mint(SfgFarm,  21000000 * 10 ** uint256(decimals));      // 100%, 21000000
 }
}

contract SfyToken is ERC20 {
```

```
    constructor(address SfyFarm) ERC20("Stable Finance Yield Token", "SFY") public {
        uint8 decimals = 18;
        _setupDecimals(decimals);

        _mint(SfyFarm,  21000000 * 10 ** uint256(decimals));      // 100%, 21000000
    }
}
pragma solidity ^0.6.0;

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

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

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

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

Knownsec

5. Appendix B: vulnerability risk rating criteria

Smart contract vulnerability rating standard	
Vulnerability rating	Vulnerability rating description
High risk vulnerability	The loophole which can directly cause the contract or the user's fund loss, such as the value overflow loophole which can cause the value of the substitute currency to zero, the false recharge loophole that can cause the exchange to lose the substitute coin, can cause the contract account to lose the ETH or the reentry loophole of the substitute currency, and so on; It can cause the loss of ownership rights of token contract, such as: the key function access control defect or call injection leads to the key function access control bypassing, and the loophole that the token contract can not work properly. Such as: a denial-of-service vulnerability due to sending ETHs to a malicious address, and a denial-of-service vulnerability due to gas depletion.
Middle risk vulnerability	High risk vulnerabilities that need specific addresses to trigger, such as numerical overflow vulnerabilities that can be triggered by the owner of a token contract, access control defects of non-critical functions, and logical design defects that do not result in direct capital losses, etc.
Low risk vulnerability	A vulnerability that is difficult to trigger, or that will harm a limited number after triggering, such as a numerical overflow that requires a large number of ETH or tokens to trigger, and a vulnerability that the attacker cannot directly profit from after triggering a numerical overflow. Rely on risks by specifying the order of transactions triggered by a high gas.

6. Appendix C： Introduction of test tool

6.1. Manticore

Manticore is a symbolic execution tool for analysis of binaries and smart contracts. It discovers inputs that crash programs via memory safety violations. Manticore records an instruction-level

Knownsec

trace of execution for each generated input and exposes programmatic access to its analysis engine via a Python API.

6.2. Oyente

Oyente is a smart contract analysis tool that Oyente can use to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and more. More conveniently, Oyente's design is modular, so this allows advanced users to implement and insert their own detection logic to check for custom attributes in their contracts.

6.3. securify.sh

Securify can verify common security issues with Ethereum smart contracts, such as transactional out-of-order and lack of input validation. It analyzes all possible execution paths of the program while fully automated. In addition, Securify has a specific language for specifying vulnerabilities. Securify can keep an eye on current security and other reliability issues.

6.4. Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

6.5. MAIAN

MAIAN is an automated tool for finding Ethereum smart contract vulnerabilities. Maian deals with the contract's bytecode and tries to establish a series of transactions to find and confirm errors.

6.6. ethersplay

Ethersplay is an EVM disassembler that contains related analysis tools.

Knownsec

6.7. ida-evm

Ida-evm is an IDA processor module for the 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 Toolkit

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