KNOWNSEC

Smart Contract Audit Report

Safe State

Security

★ ★ ★ ★ ★

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
SNestGauge Smart Contract Audit Report	V1.0	【SNestGauge-DMSJ-20200922】	Open project team

Copyright statement

目录

1. Review

The effective testing time of this report is from **September 19,2020** to **September 21,2020**. During this period, the Knownsec engineers audited the safety and regulatory aspects of **SNestGauge** smart contract code.

In this test, engineers comprehensively analyzed common vulnerabilities of smart contracts (Chapter 3) and It was not discovered risk, so it's evaluated as **Security**.

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..

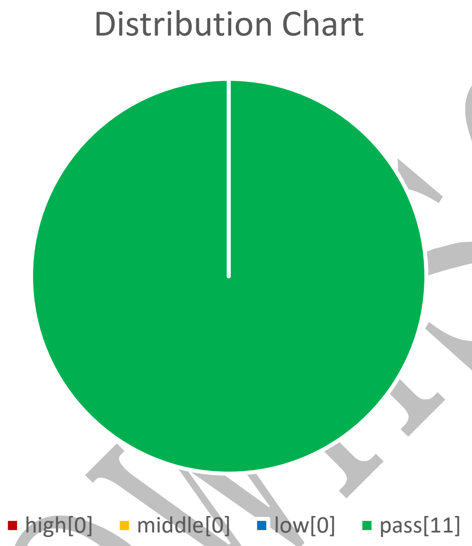
Target information for this test:

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

2. Analysis of code vulnerability

2.1. Distribution of vulnerability Levels

Vulnerability statistics			
high	Middle	low	pass
0	0	0	11



2.2. Audit result summary

Result			
Test project	Test content	status	description
Smart Contract	Reentrancy	Pass	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	Pass	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	Pass	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【Pass】

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: No related vulnerabilities in smart contract code.

Safety advice: none.

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, 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。

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【Pass】

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: No related vulnerabilities in smart contract code

Safety advice：None。

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.

Detection 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.

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

Safety advice: none

3.10. Adding tokens【Pass】

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: No related vulnerabilities in smart contract code.

Safety advice: none.

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.

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);
        }
```

```
        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;
```

```
        }

        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;
```

```
        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);
```

```
        _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]);
```

```
        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');
```

```
        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');
    }
}
*/
}
```

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 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.

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.