知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人：知道创宇区块链安全研究团队

文档信息

文档名称	CirclePool 智能合约审计报告
文档 MD5	5c0df4b9a6f2b84375e657754e4d99f4
Token 名称	智能合约
合约地址	0x4eaa907fe06667cefa77b90a703081403a0cfcc0
链接地址	https://scan.hecochain.com/address/ 0x4eaa907fe06667cefa77b90a703081403a0cfcc0#contracts
审计团队	知道创宇区块链安全研究团队
审计编号	Knownsec-bfjfznhy-0104
审计日期	2021 年 01 月 04 日
审计结果	安全

版权说明

目录

1. 综述 1

2. 代码漏洞分析 2

2.1. 漏洞等级分布 2

2.2. 审计结果汇总说明 2

3. 代码审计结果分析 4

3.1. 重入攻击检测【通过】 4

3.2. 重放攻击检测【通过】 4

3.3. 重排攻击检测【通过】 4

3.4. 数值溢出检测【通过】 4

3.5. 算术精度误差【通过】 5

3.6. 访问控制检测【通过】 5

3.7. tx.origin 身份验证【通过】 5

3.8. call 注入攻击【通过】 6

3.9. 返回值调用验证【通过】 6

3.10. 未初始化的储存指针【通过】 6

3.11. 错误使用随机数【通过】 7

3.12. 交易顺序依赖【通过】 7

3.13. 拒绝服务攻击【通过】 7

3.14. 逻辑设计缺陷【通过】 8

3.15. 假充值漏洞【通过】 8

3.16. 增发代币漏洞【通过】 8

3.17. 冻结账户绕过【通过】 8

3.18. 编译器版本安全【通过】 8

3.19. 不推荐的编码方式【通过】 9

3.20. 冗余代码【通过】 9

3.21. 安全算数库的使用【通过】 9

3.22. require/assert 的使用【通过】 9
3.23. Gas 消耗【通过】 9
3.24. fallback 函数使用【通过】 10
3.25. owner 权限控制【通过】 10
3.26. 低级函数安全【通过】 10
3.27. 变量覆盖【通过】 10
3.28. 时间戳依赖攻击【通过】 10
3.29. 不安全的接口使用【通过】 11

4. 附录A：合约代码 12

5. 附录 B：漏洞风险评级标准 57

6. 附录 C：漏洞测试工具简介 58

6.1. Manticore 58
6.2. Oyente 58
6.3. securify.sh 58
6.4. Echidna 58
6.5. MAIAN 58
6.6. ethersplay 59
6.7. ida-evm 59
6.8. Remix-ide 59
6.9. 知道创宇渗透测试人员专用工具包 59

1. 综述

本次报告有效测试时间是从 2020 年 12 月 30 日开始到 2021 年 01 月 03 日结束，在此期间针对 CirclePool 合约代码的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，未发现安全漏洞，故综合评定为安全。

本次智能合约安全审计结果：安全

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次测试的目标信息

合约名称	CirclePool 智能合约审计报告
代码类型	智能合约
代码语言	**Solidity**
合约地址	https://scan.hecochain.com/address/ 0x4eaa907fe06667cefa77b90a703081403a0cfcc0#contracts

2. 代码漏洞分析

2.1. 漏洞等级分布

本次漏洞风险按等级统计：

漏洞风险等级个数统计表			
高危	中危	低危	通过
0	0	0	29

个数
■ 高危 ■ 中危 ■ 低危 ■ 通过

2.2. 审计结果汇总说明

智能合约审计结果			
测试序号	测试内容	测试结果	测试描述
1	重入攻击检测	通过	经检测，不存在该安全问题
2	重放攻击检测	通过	经检测，不存在该安全问题
3	重排攻击检测	通过	经检测，不存在该安全问题
4	数值溢出检测	通过	经检测，不存在该安全问题
5	算数精度误差	通过	经检测，不存在该安全问题
6	访问控制缺陷检测	通过	经检测，不存在该安全问题
7	tx.progin 身份验证	通过	经检测，不存在该安全问题

8	call 注入攻击	通过	经检测，不存在该安全问题
9	返回值调用验证	通过	经检测，不存在该安全问题
10	未初始化的存储指针	通过	经检测，不存在该安全问题
11	错误使用随机数检测	通过	经检测，不存在该安全问题
12	交易顺序依赖检测	通过	经检测，不存在该安全问题。
13	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
14	逻辑设计缺陷检测	通过	经检测，不存在该安全问题。
15	假充值漏洞检测	通过	经检测，不存在该安全问题
16	增发代币漏洞检测	通过	经检测，不存在该安全问题
17	冻结账户绕过检测	通过	经检测，不存在该安全问题
18	编译器版本安全	通过	经检测，不存在该安全问题
19	不推荐的编码方式	通过	经检测，不存在该安全问题
20	冗余代码	通过	经检测，不存在该安全问题
21	安全算数库的使用	通过	经检测，不存在该安全问题
22	require/assert 的使用	通过	经检测，不存在该安全问题
23	Gas 消耗	通过	经检测，不存在该安全问题
24	fallback 函数的使用	通过	经检测，不存在该安全问题
25	owner 权限控制	通过	经检测，不存在该安全问题
26	低级函数安全	通过	经检测，不存在该安全问题
27	变量覆盖	通过	经检测，不存在该安全问题
28	时间戳依赖攻击	通过	经检测，不存在该安全问题
29	不安全的接口使用	通过	经检测，不存在该安全问题

3. 代码审计结果分析

3.1. 重入攻击检测【通过】

重入漏洞是最著名的智能合约漏洞。

在 Solidity 中，调用其他地址的函数或者给合约地址转账，都会把自己的地址作为被调合约的 msg.sender 传递过去。此时，如果传递的 gas 足够的话，就可能会被对方进行重入攻击，在 hecochain 中对合约地址调用 transfer/sender，只会传递 2300 的 Gas，这不足以发起一次重入攻击。所以要一定避免通过 call.value 的方式，调用未知的合约地址。因为 call.value 可以传入远大于 2300 的 gas。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.2. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

3.3. 重排攻击检测【通过】

重排攻击是指矿工或其他方式试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果:经检测，智能合约代码中不存在相关洞。

安全建议:无。

3.4. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.5. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20, 而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.6. 访问控制检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。。

安全建议：无。

3.8. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

3.9. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于：transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300Gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300Gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 Gas 进行调用（可通过传入 Gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.10. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.11. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.12. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于 hecochain 区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.13. 拒绝服务攻击【通过】

在 hecochain 的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.14. 逻辑设计缺陷【通过】

检查智能合约代码中与业务设计相关的安全问题。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.15. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender]＜value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.16. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.17. 冻结账户绕过【通过】

检查代币合约中在转移代币时，是否存在未校验代币来源账户、发起账户、目标账户是否被冻结的操作。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.18. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.8 以上，不存在该安全问题。

安全建议：无。

3.19. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.20. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.21. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

3.22. require/assert 的使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.23. Gas 消耗【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.24. fallback 函数使用【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.25. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.26. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞 call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.27. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.28. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。检查合约代码实现中是否存在有依赖于时间戳的关键功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.29. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4. 附录 A：合约代码

本次测试代码来源：

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

pragma solidity ^0.6.0;

/**

 * @dev Standard math utilities missing in the Solidity language.

 */

library Math {

    /**

     * @dev Returns the largest of two numbers.

     */

    function max(uint256 a, uint256 b) internal pure returns (uint256) {

        return a >= b ? a : b;

    }

    /**

     * @dev Returns the smallest of two numbers.

     */

    function min(uint256 a, uint256 b) internal pure returns (uint256) {

        return a < b ? a : b;

    }

    /**

     * @dev Returns the average of two numbers. The result is rounded towards

     * zero.

     */

    function average(uint256 a, uint256 b) internal pure returns (uint256) {

        // (a + b) / 2 can overflow, so we distribute

        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);

    }
```

```
}
/**
  * @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 {    //knownsec //不存在整数溢出风险
    /**
      * @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");
    }
    /**
     * @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) {
    // Solidity only automatically asserts when dividing by 0
    require(b > 0, errorMessage);
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold

    return c;
}

/**
  * @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;
  }
}
```

```
/**
  * @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) {
              // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
              // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
              // for accounts without code, i.e. `keccak256('')`
              bytes32 codehash;
              bytes32                        accountHash                        =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @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 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);

        /**
          * @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);
}

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;


    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }


    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, 
to, value));
    }


    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, 
value));
    }


    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, 
newAllowance));
    }


    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value,
"SafeERC20: decreased allowance below zero");
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }


    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //  1. The target address is checked to verify it contains contract code
        //  2. The call itself is made, and success asserted
        //  3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not
succeed");
            }
        }
}



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

  /**
   * @dev Indicates that the contract has been initialized.
   */
  bool private initialized;

  /**
   * @dev Indicates that the contract is in the process of being initialized.
   */
  bool private initializing;
  /**
   * @dev Modifier to use in the initializer function of a contract.
   */
  modifier initializer() {
    require(initializing || isConstructor() || !initialized, "Contract instance has already been initialized");

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

    _;

    if (isTopLevelCall) {
      initializing = false;
    }
  }

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

```
        return cs == 0;
    }


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




/**
  * @dev Contract module that helps prevent reentrant calls to a function.
  *
  * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
  * available, which can be applied to functions to make sure there are no nested
  * (reentrant) calls to them.
  *
  * Note that because there is a single `nonReentrant` guard, functions marked as
  * `nonReentrant` may not call one another. This can be worked around by making
  * those functions `private`, and then adding `external` `nonReentrant` entry
  * points to them.
  *
  * TIP: If you would like to learn more about reentrancy and alternative ways
  * to protect against it, check out our blog post
  * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
  */
contract ReentrancyGuardUpgradeSafe is Initializable {
        bool private _notEntered;



        function __ReentrancyGuard_init() internal initializer {
                __ReentrancyGuard_init_unchained();
```

```
}

function __ReentrancyGuard_init_unchained() internal initializer {


        // Storing an initial non-zero value makes deployment a bit more
        // expensive, but in exchange the refund on every call to nonReentrant
        // will be lower in amount. Since refunds are capped to a percetange of
        // the total transaction's gas, it is best to keep them low in cases
        // like this one, to increase the likelihood of the full refund coming
        // into effect.
        _notEntered = true;

}


/**
  * @dev Prevents a contract from calling itself, directly or indirectly.
  * Calling a `nonReentrant` function from another `nonReentrant`
  * function is not supported. It is possible to prevent this from happening
  * by making the `nonReentrant` function external, and make it call a
  * `private` function that does the actual work.
  */
modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_notEntered, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _notEntered = false;
        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _notEntered = true;
    }

    uint256[49] private __gap;
}


contract Governable is Initializable {
    address public governor;

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

    /**
     * @dev Contract initializer.
     * called once by the factory at time of deployment
     */
    function initialize(address governor_) virtual public initializer { //knownsec //初始化
        governor = governor_;
        emit GovernorshipTransferred(address(0), governor);
    }

    modifier governance() { //knownsec //权限验证
        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 { //knownsec //销毁合约
    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 {   //knownsec //转
让合约
    _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)); //knownsec //验证合约地址是否为空地址，防止
转丢
    emit GovernorshipTransferred(governor, newGovernor);
    governor = newGovernor;
}
}


contract Configurable is Governable {     //knownsec //配置相关

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) public governance {
    _setConfig(bytes32(uint(key) ^ uint(addr)), value);
}
}


contract RewardsDistributor is Configurable {
using SafeERC20 for IERC20;

address public rewardsToken;

function initialize(address governor, address _rewardsToken) public initializer { //knownsec //
初始化
    super.initialize(governor);
    rewardsToken = _rewardsToken;
}

function approvePool(address pool, uint amount) public governance {
    //IERC20(rewardsToken).safeApprove(pool, amount);
    IERC20(rewardsToken).approve(pool, amount);                    // GT do not support
safeApprove
}
}
```

```
// Inheritancea
interface IStakingRewards {
    // Views
    function lastTimeRewardApplicable() external view returns (uint256);


    function rewardPerToken() external view returns (uint256);


    function rewards(address account) external view returns (uint256);


    function earned(address account) external view returns (uint256);


    function getRewardForDuration() external view returns (uint256);


    function totalSupply() external view returns (uint256);


    function balanceOf(address account) external view returns (uint256);


    // Mutative


    function stake(uint256 amount) external;


    function withdraw(uint256 amount) external;


    function getReward() external;


    function exit() external;
}


abstract contract RewardsDistributionRecipient {
    address public rewardsDistribution;
    function notifyRewardAmount(uint256 reward) virtual external;


    modifier onlyRewardsDistribution() {
        require(msg.sender == rewardsDistribution, "Caller is not RewardsDistribution contract");
        _;
    }
}


contract StakingRewards is IStakingRewards, RewardsDistributionRecipient, ReentrancyGuardUpgradeSafe {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;


    /* ========== STATE VARIABLES ========== */


    IERC20 public rewardsToken;
    IERC20 public stakingToken;
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;                      // obsoleted
    uint256 public rewardsDuration = 60 days;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;


    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) override public rewards;


    uint256 internal _totalSupply;
    mapping(address => uint256) internal _balances;
```

```
/* ========== CONSTRUCTOR ========== */

//constructor(
function initialize(
        address _rewardsDistribution,
        address _rewardsToken,
        address _stakingToken
) public virtual initializer {
        super.__ReentrancyGuard_init();
        rewardsToken = IERC20(_rewardsToken);
        stakingToken = IERC20(_stakingToken);
        rewardsDistribution = _rewardsDistribution;
}

/* ========== VIEWS ========== */

function totalSupply() virtual override public view returns (uint256) {
        return _totalSupply;
}

function balanceOf(address account) virtual override public view returns (uint256) {
        return _balances[account];
}

function lastTimeRewardApplicable() override public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
}

function rewardPerToken() virtual override public view returns (uint256) {
        if (_totalSupply == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
lastTimeRewardApplicable().sub(lastUpdateTime).mul(rewardRate).mul(1e18).div(_totalSupply)
            );
    }

    function earned(address account) virtual override public view returns (uint256) {
        return
_balances[account].mul(rewardPerToken().sub(userRewardPerTokenPaid[account])).div(1e18).a
dd(rewards[account]);
    }

    function getRewardForDuration() virtual override external view returns (uint256) {
        return rewardRate.mul(rewardsDuration);
    }

    /* ========== MUTATIVE FUNCTIONS ========== */

    function stakeWithPermit(uint256 amount, uint deadline, uint8 v, bytes32 r, bytes32 s) virtual
public nonReentrant updateReward(msg.sender) {
        require(amount > 0, "Cannot stake 0");
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);

        // permit
        IUniswapV2ERC20(address(stakingToken)).permit(msg.sender, address(this), amount,
deadline, v, r, s);
```

```
        stakingToken.safeTransferFrom(msg.sender, address(this), amount);
        emit Staked(msg.sender, amount);
    }

    function stake(uint256 amount) virtual override public nonReentrant
updateReward(msg.sender) {
        require(amount > 0, "Cannot stake 0");
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        stakingToken.safeTransferFrom(msg.sender, address(this), amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) virtual override public nonReentrant
updateReward(msg.sender) {
        require(amount > 0, "Cannot withdraw 0");
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        stakingToken.safeTransfer(msg.sender, amount);
        emit Withdrawn(msg.sender, amount);
    }

    function getReward() virtual override public nonReentrant updateReward(msg.sender) {
        uint256 reward = rewards[msg.sender];
        if (reward > 0) {
            rewards[msg.sender] = 0;
            rewardsToken.safeTransfer(msg.sender, reward);
            emit RewardPaid(msg.sender, reward);
        }
    }

    function exit() virtual override public {
        withdraw(_balances[msg.sender]);
        getReward();
    }

    /* ========== RESTRICTED FUNCTIONS ========== */

    function notifyRewardAmount(uint256 reward) override external onlyRewardsDistribution
updateReward(address(0)) {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(rewardsDuration);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(rewardsDuration);
        }

        // Ensure the provided reward amount is not more than the balance in the contract.
        // This keeps the reward rate in the right range, preventing overflows due to
        // very high values of rewardRate in the earned and rewardsPerToken functions;
        // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
        uint balance = rewardsToken.balanceOf(address(this));
        require(rewardRate <= balance.div(rewardsDuration), "Provided reward too high");

        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(rewardsDuration);
        emit RewardAdded(reward);
    }
    /* ========== MODIFIERS ========== */

    modifier updateReward(address account) virtual {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    /* ========== EVENTS ========== */

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);
}

interface IUniswapV2ERC20 {
    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external;
}

contract StakingPool is Configurable, StakingRewards {
    using Address for address payable;

    bytes32 internal constant _ecoAddr_            = 'ecoAddr';
    bytes32 internal constant _ecoRatio_           = 'ecoRatio';
    bytes32 internal constant _allowContract_      = 'allowContract';
    bytes32 internal constant _allowlist_          = 'allowlist';
    bytes32 internal constant _blocklist_          = 'blocklist';

    uint public lep;                  // 1: linear, 2: exponential, 3: power
    uint public period;
    uint public begin;

    mapping (address => uint256) public paid;

    function initialize(address _governor,
        address _rewardsDistribution,
        address _rewardsToken,
        address _stakingToken,
        address _ecoAddr
    ) public virtual initializer {    //knownsec //初始化
        super.initialize(_governor);
        super.initialize(_rewardsDistribution, _rewardsToken, _stakingToken);
        config[_ecoAddr_] = uint(_ecoAddr);
        config[_ecoRatio_] = 0.1 ether;
    }

    function notifyRewardBegin(uint _lep, uint _period, uint _span, uint _begin) virtual public
governance updateReward(address(0)) {
        lep             = _lep;         // 1: linear, 2: exponential, 3: power
        period          = _period;
        rewardsDuration = _span;
        begin           = _begin;
        periodFinish    = _begin.add(_span);
    }

    function rewardDelta() public view returns (uint amt) {
        if(begin == 0 || begin >= now || lastUpdateTime >= now)
            return 0;

        amt = rewardsToken.allowance(rewardsDistribution, address(this)).sub(rewards[address(0)]);

        // calc rewardDelta in period
        if(lep == 3) {                                         // power
            uint y = period.mul(1 ether).div(lastUpdateTime.add(rewardsDuration).sub(begin));
            uint amt1 = amt.mul(1 ether).div(y);
            uint amt2 = amt1.mul(period).div(now.add(rewardsDuration).sub(begin));
            amt = amt.sub(amt2);
        } else if(lep == 2) {                                  // exponential
            if(now.sub(lastUpdateTime) < rewardsDuration)
                amt = amt.mul(now.sub(lastUpdateTime)).div(rewardsDuration);
        }else if(now < periodFinish)                           // linear
            amt = amt.mul(now.sub(lastUpdateTime)).div(periodFinish.sub(lastUpdateTime));
        else if(lastUpdateTime >= periodFinish)
            amt = 0;
    }

    function rewardPerToken() virtual override public view returns (uint256) {
        if (_totalSupply == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                rewardDelta().mul(1e18).div(_totalSupply)
            );
    }

    modifier updateReward(address account) virtual override {
        (uint delta, uint d) = (rewardDelta(), 0);
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = now;
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }

        address addr = address(config[_ecoAddr_]);
        uint ratio = config[_ecoRatio_];
        if(addr != address(0) && ratio != 0) {
            d = delta.mul(ratio).div(1 ether);
            rewards[addr] = rewards[addr].add(d);
        }
        rewards[address(0)] = rewards[address(0)].add(delta).add(d);
        _;
    }

    function getReward() virtual override public {
        getReward(msg.sender);
    }
    function getReward(address payable acct) virtual public nonReentrant updateReward(acct) {
        require(acct != address(0), 'invalid address'); //knownsec //验证是否为空地址
        require(getConfig(_blocklist_, acct) == 0, 'In blocklist'); //knownsec // 黑名单
        bool isContract = acct.isContract();
        require(!isContract || config[_allowContract_] != 0 || getConfig(_allowlist_, acct) != 0,
'No allowContract');

        uint256 reward = rewards[acct];
        if (reward > 0) {
            paid[acct] = paid[acct].add(reward);
            paid[address(0)] = paid[address(0)].add(reward);
            rewards[acct] = 0;
            rewards[address(0)] = rewards[address(0)] > reward ?
rewards[address(0)].sub(reward) : 0;
            rewardsToken.safeTransferFrom(rewardsDistribution, acct, reward);
            emit RewardPaid(acct, reward);
        }
    }

    function getRewardForDuration() override external view returns (uint256) {
        return rewardsToken.allowance(rewardsDistribution,
address(this)).sub(rewards[address(0)]);
    }

}

contract DoublePool is StakingPool {
    IStakingRewards public stakingPool2;
    IERC20 public rewardsToken2;
    //uint256 public lastUpdateTime2;                                 // obsoleted
    //uint256 public rewardPerTokenStored2;                           // obsoleted
    mapping(address => uint256) public userRewardPerTokenPaid2;
    mapping(address => uint256) public rewards2;


    function initialize(address, address, address, address, address) override public {
        revert();
    }


    function initialize(address _governor, address _rewardsDistribution, address _rewardsToken,
address _stakingToken, address _ecoAddr, address _stakingPool2, address _rewardsToken2) public
initializer {
        super.initialize(_governor, _rewardsDistribution, _rewardsToken, _stakingToken,
_ecoAddr);   //knownsec //初始化


        stakingPool2 = IStakingRewards(_stakingPool2);
        rewardsToken2 = IERC20(_rewardsToken2);
    }


    function notifyRewardBegin(uint _lep, uint _period, uint _span, uint _begin) virtual override
public governance updateReward2(address(0)) {
        super.notifyRewardBegin(_lep, _period, _span, _begin);
    }


    function stake(uint amount) virtual override public updateReward2(msg.sender) {
        super.stake(amount);
        stakingToken.safeApprove(address(stakingPool2), amount);
        stakingPool2.stake(amount);
    }


    function withdraw(uint amount) virtual override public updateReward2(msg.sender) {
        stakingPool2.withdraw(amount);
        super.withdraw(amount);
    }
    function getReward2() virtual public nonReentrant updateReward2(msg.sender) {    //knownsec //奖励
        uint256 reward2 = rewards2[msg.sender];
        if (reward2 > 0) {
            rewards2[msg.sender] = 0;
            stakingPool2.getReward();
            rewardsToken2.safeTransfer(msg.sender, reward2);
            emit RewardPaid2(msg.sender, reward2);
        }
    }
    event RewardPaid2(address indexed user, uint256 reward2);

    function getDoubleReward() virtual public {
        getReward();
        getReward2();
    }

    function exit() override public {
        super.exit();
        getReward2();
    }

    function rewardPerToken2() virtual public view returns (uint256) {
        return stakingPool2.rewardPerToken();
    }

    function earned2(address account) virtual public view returns (uint256) {
        return
_balances[account].mul(rewardPerToken2().sub(userRewardPerTokenPaid2[account])).div(1e18)
.add(rewards2[account]);
```

```
    }

    modifier updateReward2(address account) virtual {
        if (account != address(0)) {
            rewards2[account] = earned2(account);
            userRewardPerTokenPaid2[account] = rewardPerToken2();
        }
        _;
    }

}

contract CirclePool is StakingPool {
    bytes32 internal constant _stakingThreshold_         = 'stakingThreshold';
    bytes32 internal constant _refererWeight_            = 'refererWeight';

    ICircle public circle;
    uint256 public totalSupplyRefer;
    uint256 public totalSupplyCircle;
    mapping (address => uint256) public balanceReferOf;
    mapping (uint256 => uint256) public balanceOfCircle;
    mapping (uint256 => uint256) public supplyOfCircle;
    mapping (uint256 => uint256) public rewardPerTokenStoredCircle;
    mapping (address => mapping (uint256 => uint256)) public userRewardPerTokenCircle;
// acct => tokenID => rewardPerToken
    mapping (address => uint256) public eligible;
    mapping (uint256 => uint256) public eligibleCount;

    function __CirclePool_init(
        address _governor,
        address _rewardsDistribution,
        address _rewardsToken,
        address _stakingToken,
        address _ecoAddr,
        address circle_
    ) public virtual initializer {
        super.initialize(_governor, _rewardsDistribution, _rewardsToken, _stakingToken, _ecoAddr);
        __CirclePool_init_unchained(circle_);
    }

    function __CirclePool_init_unchained(address circle_) public governance {
        circle = ICircle(circle_);
        config[_stakingThreshold_]      = 100 ether;
        _setConfig(_refererWeight_, 1, 0.25 ether);
        _setConfig(_refererWeight_, 2, 0.10 ether);
    }

    function amendSupplyOfCircle(uint id) external {
        uint oldSupply = supplyOfCircle[id];
        uint supply;
        for(uint i=0; i<circle.membersCount(id); i++)     //knownsec //循环过大可能会导致gas 耗尽，正常情况下几乎无影响
            supply = supply.add(balanceCircleOf(circle.members(id, i)));
        supplyOfCircle[id] = supply;
        totalSupplyCircle = totalSupplyCircle.add(supply).sub(oldSupply);
    }

    function totalSupplySum() virtual public view returns (uint) {
        return totalSupply().add(totalSupplyRefer).add(totalSupplyCircle);
    }


    function balanceCircleOf(address acct) virtual public view returns (uint) {
        if(eligible[acct] == uint(-1) || eligible[acct] == 0 && lptNetValue(_balances[acct]) <
config[_stakingThreshold_])
            return 0;
        else if(circle.balanceOf(acct) > 0)
            return                      balanceOfCircle[circle.tokenOfOwnerByIndex(acct,
0)].sub(_balances[acct]);
        else if(circle.circleOf(acct) != 0)
            return
balanceOfCircle[circle.circleOf(acct)].div(circle.membersCount(circle.circleOf(acct)));
    }
    function balanceSumOf(address acct) virtual public view returns (uint) {
        return balanceOf(acct).add(balanceReferOf[acct]).add(balanceCircleOf(acct));
    }


    function lptNetValue(uint vol) public view returns (uint) {
        if(vol == 0)
            return 0;
        CircleSwapRouter03 router = CircleSwapRouter03(circle.router());
        address WHT = router.WETH();
        uint wht = IERC20(WHT).balanceOf(address(stakingToken));
        if(wht > 0) {
            return wht.mul(vol).div(IERC20(stakingToken).totalSupply()).mul(2);
        } else {
            uint cir = IERC20(rewardsToken).balanceOf(address(stakingToken));
            //require(cir > 0);
            cir = cir.mul(vol).div(IERC20(stakingToken).totalSupply()).mul(2);
            (uint          reserve0,          uint          reserve1,)          =
IUniswapV2Pair(IUniswapV2Factory(router.factory()).getPair(WHT,
address(rewardsToken))).getReserves();
            //(reserve0, reserve1) = tokenA == WHT < rewardsToken ? (reserve0, reserve1) : (reserve1, reserve0);
            return WHT < address(rewardsToken) ? cir.mul(reserve0) / reserve1 : cir.mul(reserve1) / reserve0;
        }
    }

    function _updateEligible(address acct) internal {
        uint oldEligible = eligible[acct];
        eligible[acct] = lptNetValue(_balances[acct]) >= config[_stakingThreshold_] ? 1 : uint(-1);
        uint id = circle.circleOf(acct);
        if(id != 0) {
            if(oldEligible != 1 && eligible[acct] == 1)
                eligibleCount[id] = eligibleCount[id].add(1);
            else if(oldEligible == 1 && eligible[acct] == uint(-1))
                eligibleCount[id] = eligibleCount[id].sub(1);
        }
    }

    function _increaseBalanceRefer(address referee, uint increasement) internal {
        address referer   = circle.refererOf(referee);
        address referer2 = circle.refererOf(referer);
        uint inc1 = circleOf(referer)   != 0 ? increasement.mul(getConfig(_refererWeight_, 1)).div(1 ether) : 0;
        uint inc2 = circleOf(referer2) != 0 ? increasement.mul(getConfig(_refererWeight_, 2)).div(1 ether) : 0;
        balanceReferOf[referer]   = balanceReferOf[referer ].add(inc1);
        balanceReferOf[referer2] = balanceReferOf[referer2].add(inc2);
        totalSupplyRefer          = totalSupplyRefer.add(inc1).add(inc2);
    }

    function _decreaseBalanceRefer(address referee, uint decreasement) internal {
        address referer   = circle.refererOf(referee);
        address referer2 = circle.refererOf(referer);
        uint dec1 = circleOf(referer)    != 0 ? decreasement.mul(getConfig(_refererWeight_, 1)).div(1 ether) : 0;
        uint dec2 = circleOf(referer2)   != 0 ? decreasement.mul(getConfig(_refererWeight_, 2)).div(1 ether) : 0;
        balanceReferOf[referer]   = balanceReferOf[referer ].sub(dec1);
        balanceReferOf[referer2] = balanceReferOf[referer2].sub(dec2);
        totalSupplyRefer          = totalSupplyRefer.sub(dec1).sub(dec2);
    }

    function _increaseBalanceCircle(address acct, uint increasement, uint oldBalanceCircle) internal {
        uint id = circleOf(acct);
        if(id == 0)
            return;
        balanceOfCircle[id] = balanceOfCircle[id].add(increasement);
        uint newBalanceCircle = balanceCircleOf(acct);
        uint delta = _deltaSupplyCircle(id, acct, increasement).add(newBalanceCircle).sub(oldBalanceCircle);
        supplyOfCircle[id] = supplyOfCircle[id].add(delta);
        totalSupplyCircle = totalSupplyCircle.add(delta);
    }

    function _decreaseBalanceCircle(address acct, uint decreasement, uint oldBalanceCircle) internal {
        uint id = circleOf(acct);
        if(id == 0)
```

```
            return;
        balanceOfCircle[id] = balanceOfCircle[id].sub(decreasement);
        uint newBalanceCircle = balanceCircleOf(acct);
        uint delta = _deltaSupplyCircle(id, acct, decreasement).add(oldBalanceCircle).sub(newBalanceCircle);
        supplyOfCircle[id] = supplyOfCircle[id].sub(delta);
        totalSupplyCircle = totalSupplyCircle.sub(delta);
    }

    function _deltaSupplyCircle(uint id, address acct, uint deltaBalance) internal view returns (uint delta) {
        delta = deltaBalance.mul(eligibleCount[id]).div(circle.membersCount(id));
        if(circle.circleOf(acct) != 0) {
            if(eligible[acct] == 1)
                delta = delta.sub(deltaBalance.div(circle.membersCount(id)));
            if(eligible[circle.ownerOf(id)] == 1)
                delta = delta.add(deltaBalance);
        }
    }

    function circleOf(address acct) public view returns (uint id) {
        id = circle.circleOf(acct);
        if(id == 0 && circle.balanceOf(acct) > 0)
            id = circle.tokenOfOwnerByIndex(acct, 0);
    }

    function stakeWithPermit(uint256 amount, uint deadline, uint8 v, bytes32 r, bytes32 s) virtual override public {
        require(circleOf(msg.sender) != 0, "Not in circle");      //knownsec //id 必须不为 0
        uint balanceCircle = balanceCircleOf(msg.sender);
        super.stakeWithPermit(amount, deadline, v, r, s);
        _updateEligible(msg.sender);
        _increaseBalanceRefer(msg.sender, amount);
        _increaseBalanceCircle(msg.sender, amount, balanceCircle);
    }

    function stake(uint256 amount) virtual override public {
        require(circleOf(msg.sender) != 0, "Not in circle");
        uint balanceCircle = balanceCircleOf(msg.sender);
        super.stake(amount);
        _updateEligible(msg.sender);
        _increaseBalanceRefer(msg.sender, amount);
        _increaseBalanceCircle(msg.sender, amount, balanceCircle);
    }

    function withdraw(uint256 amount) virtual override public {
        uint balanceCircle = balanceCircleOf(msg.sender);
        super.withdraw(amount);
        _updateEligible(msg.sender);
        _decreaseBalanceRefer(msg.sender, amount);
        _decreaseBalanceCircle(msg.sender, amount, balanceCircle);
    }

    function rewardPerToken() override public view returns (uint256) {
        if (_totalSupply == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                rewardDelta().mul(1e18).div(totalSupplySum())
            );
    }

    function rewardPerTokenCircle(uint id) virtual public view returns (uint) {
        if (supplyOfCircle[id] == 0) {
            return rewardPerTokenStoredCircle[id];
        }
        return
            rewardPerTokenStoredCircle[id].add(
                rewardDeltaCircle(id).mul(1e18).div(supplyOfCircle[id])
            );
    }

    function rewardDeltaCircle(uint id) virtual public view returns (uint) {
        return
supplyOfCircle[id].mul(rewardPerToken().sub(userRewardPerTokenPaid[address(id)])).div(1e18)
;
    }

    function earned(address acct) override public view returns (uint256) {
        uint id = circleOf(acct);
        return
balanceCircleOf(acct).mul(rewardPerTokenCircle(id).sub(userRewardPerTokenCircle[acct][id])).
div(1e18).add(

balanceOf(acct).add(balanceReferOf[acct]).mul(rewardPerToken().sub(userRewardPerTokenPaid
[acct])).div(1e18).add(rewards[acct]));
    }

    modifier updateReward(address acct) virtual override {
        (uint delta, uint d) = (rewardDelta(), 0);

        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = now;
        if (acct != address(0)) {
            if(eligible[acct] == 0)
                _updateEligible(acct);

            _updateReward(acct);

            uint id = circleOf(acct);
            if(id != 0) {
                userRewardPerTokenCircle[acct][id] = rewardPerTokenStoredCircle[id] =
rewardPerTokenCircle(id);
                userRewardPerTokenPaid[address(id)] = rewardPerTokenStored;
            }

            _updateReward(acct = circle.refererOf(acct));
            _updateReward(circle.refererOf(acct));
        }

        address addr = address(config[_ecoAddr_]);
        uint ratio = config[_ecoRatio_];
        if(addr != address(0) && ratio != 0) {
            d = delta.mul(ratio).div(1 ether);
            rewards[addr] = rewards[addr].add(d);
        }
        rewards[address(0)] = rewards[address(0)].add(delta).add(d);

        _;
    }

    function _updateReward(address acct) virtual internal {
        rewards[acct] = earned(acct);

        userRewardPerTokenPaid[acct] = rewardPerTokenStored;

    }


}


interface ICircle {              // is IERC721, IERC721Metadata, IERC721Enumerable {

    function ownerOf(uint256 tokenId) external view returns (address owner);

    function balanceOf(address owner) external view returns (uint256 balance);

    function tokenOfOwnerByIndex(address owner, uint256 index) external view returns (uint256
tokenId);


    function router() external view returns (address);

    function refererOf(address) external view returns (address);

    function circleOf(address) external view returns (uint);

    function members(uint256 tokenID, uint i) external view returns (address);

    function membersCount(uint) external view returns (uint);

}


interface CircleSwapRouter03 {

    function WETH() external pure returns (address);

    function factory() external pure returns (address);

}


interface IUniswapV2Factory {

    function getPair(address tokenA, address tokenB) external view returns (address pair);

}


interface IUniswapV2Pair {

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

5. 附录 B：漏洞风险评级标准

智能合约漏洞评级标准	
漏洞评级	漏洞评级说明
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失ETH或代币的重入漏洞等；能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call注入导致关键函数访问控制绕过等；能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送ETH导致的拒绝服务漏洞、因gas耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量ETH或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高gas触发的事务顺序依赖风险等

6. 附录 C：漏洞测试工具简介

6.1. Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

6.2. Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

6.3. securify.sh

Securify 可以验证 hecochain 智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

6.4. Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

6.5. MAIAN

MAIAN 是一个用于查找 hecochain 智能合约漏洞的自动化工具，Maian 处理合约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6. ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7. ida-evm

ida-evm 是一个针对 hecochain 虚拟机（EVM）的 IDA 处理器模块。

6.8. Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建 hecochain 合约并调试交易。

6.9. 知道创宇渗透测试人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话	+86(10)400 161 0866
邮　　箱	bfjfzx@knownsec.com
官　　网	www.knownsec.com
地　　址	北京市 朝阳区 望京SOHO T3-A座-15层