# 智能合约审计报告

安全状态

## 安全

★ ★ ★ ★ ★

# 版本说明

| 修订内容 | 修订时间 | 版本号 |
|---|---|---|
| 编写文档 | 2020/9/15 | V1.0 |

# 文档信息

| 文档名称 | 文档版本号 | 文档编号 | 保密级别 |
|---|---|---|---|
| 智能合约审计报告 | V1.0 | 【JGN-20200914】 | 项目组公开 |

# 声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

# 目录

# 1. 综述

本次报告有效测试时间是从 2020 年 9 月 15 日开始到 2020 年 9 月 16 日结束，在此期间针对**智能合约代码**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，综合评定为通过。

## 本次智能合约安全审计结果： 通过

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：**

| 模块名称 | |
|---|---|
| 代码类型 | 代币代码 |
| 代码语言 | solidity |
| 合约地址 | 0xD0d0a97DE14D676eb2e2bC9A007e9b191D478f4F |
| 合约链接 | https://etherscan.io/address/0xD0d0a97DE14D676eb2e2bC9A007e9b191D478f4F |

# 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 | 11 |

风险等级分布图
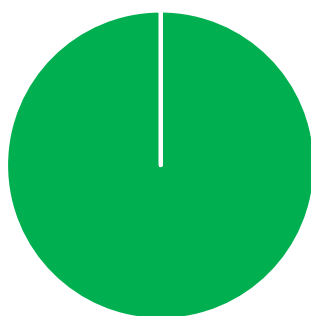


■高危[0个]　■中危[0个]　■低危[0个]　■通过[11个]

## 2.2. 审计结果汇总说明

| | 审计结果 | | |
|---|---|---|---|
| 测试项目 | 测试内容 | 状态 | 描述 |
| 智能合约 | 重入攻击检测 | 通过 | 经检测，不存在该安全问题。 |
| | 数值溢出检测 | 通过 | 经检测，不存在该安全问题。 |
| | 访问控制缺陷检测 | 通过 | 经检测，不存在该安全问题。 |
| | 未验证返回值的调用 | 通过 | 经检测，不存在该安全问题。 |
| | 错误使用随机数检测 | 通过 | 经检测，不存在该安全问题。 |
| | 事务顺序依赖检测 | 通过 | 经检测，不存在该安全问题。 |
| | 拒绝服务攻击检测 | 通过 | 经检测，不存在该安全问题。 |
| | 逻辑设计缺陷检测 | 通过 | 经检测，不存在该安全问题。 |
| | 假充值漏洞检测 | 通过 | 经检测，不存在该安全问题。 |
| | 增发代币漏洞检测 | 通过 | 经检测，不存在该安全问题。 |
| | 冻结账户绕过检测 | 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以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

**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. 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。

## 3.3. **访问控制检测**【通过】

访问控制缺陷是所有程序中都可能存在的安全风险，智能合约也同样会存在类似问题，著名的 Parity Wallet 智能合约就受到过该问题的影响。

**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。

## 3.4. **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。

## 3.5. **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。

## 3.6. 事务顺序依赖【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。由于以太坊区块链是公开的，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相关漏洞。

**安全建议：** 无

## 3.7. 拒绝服务攻击【通过】

在以太坊的世界中，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。

## 3.8. **逻辑设计缺陷**【通过】

检测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。

## 3.9. **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。

## 3.10. **增发代币漏洞**【通过】

检测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问题。

**安全建议：** 无

## 3.11. **冻结账户绕过**【通过】

检测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。

**安全建议：** 无。

# 4. 附录 A：合约代码

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-14
*/

/*
   ____                     __        __        __
  / ___/__  ___  __/ /_  / /_ ___ / /_ (_)__  __
 _\ \ / // // __ \/ __// _ \/ -_)/ __// // \ \/
/___/ \_, //_//_/\__//_//_/\__/ \__//_/ /_\_\
     /___/

* Synthetix: JGNRewards.sol
*
* Docs: https://docs.synthetix.io/
*
*
* MIT License
* ===========
*
* Copyright (c) 2020 Synthetix
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all
* copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
*/

// File: @openzeppelin/contracts/math/Math.sol

pragma solidity ^0.5.0;

/**
 * @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);
    }
}
```

```
// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

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

        return c;
    }

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

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom me
ssage on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure retur
ns (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.
     *
     * _Available since v2.4.0._
     */
    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.
      *
      * _Available since v2.4.0._
      */
     function mod(uint256 a, uint256 b, string memory errorMessage) internal pure retur
ns (uint256) {
         require(b != 0, errorMessage);
         return a % b;
     }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

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

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }
```

```
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing 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.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-
yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is r
```

```
eturned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8
045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @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].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @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 co
ntract,
 * 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) i
nternal {
```

```
        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, spend
er, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) inter
nal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spend
er, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) inter
nal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "Saf
eERC20: decreased allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spend
er, 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 retur
ned 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 no
t succeed");
        }
    }
}

// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;

    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution
");
```

```solidity
            _;
    }

    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;




contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public y = IERC20(0x73374Ea518De7adDD4c2B624C0e8B113955ee041);

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

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

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

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        y.safeTransferFrom(msg.sender, address(this), amount);
    }

    function withdraw(uint256 amount) public {
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        y.safeTransfer(msg.sender, amount);
    }
}

contract JGNRewards is LPTokenWrapper, IRewardDistributionRecipient {
    IERC20 public jgn = IERC20(0x73374Ea518De7adDD4c2B624C0e8B113955ee041);
    uint256 public constant DURATION = 7 days;

    uint256 public initreward = 10000*1e18;
    uint256 public starttime = 1600185600; //utc+8 2020 09-16 00:00:00
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;

    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);

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

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

function rewardPerToken() 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) public view returns (uint256) {
    return
        balanceOf(account)
            .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
            .div(1e18)
            .add(rewards[account]);
}

// stake visibility is public as overriding LPTokenWrapper's stake() function
function stake(uint256 amount) public updateReward(msg.sender) checkStart{
    require(amount > 0, "Cannot stake 0");
    super.stake(amount);
    emit Staked(msg.sender, amount);
}

function withdraw(uint256 amount) public updateReward(msg.sender) checkStart{
    require(amount > 0, "Cannot withdraw 0");
    super.withdraw(amount);
    emit Withdrawn(msg.sender, amount);
}

function exit() external {
    withdraw(balanceOf(msg.sender));
    getReward();
}

function getReward() public updateReward(msg.sender) checkStart{
    uint256 reward = earned(msg.sender);
    if (reward > 0) {
        rewards[msg.sender] = 0;
        jgn.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}

modifier checkStart(){
    require(block.timestamp > starttime,"not start");
    _;
}

function notifyRewardAmount(uint256 reward)
    external
    onlyRewardDistribution
    updateReward(address(0))
{
    if (block.timestamp >= periodFinish) {
        rewardRate = reward.div(DURATION);
    } else {
        uint256 remaining = periodFinish.sub(block.timestamp);
        uint256 leftover = remaining.mul(rewardRate);
        rewardRate = reward.add(leftover).div(DURATION);
    }
    lastUpdateTime = block.timestamp;
    periodFinish = block.timestamp.add(DURATION);
    emit RewardAdded(reward);
}
}
```

# 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；<br>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；<br>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 包含一个符号以太坊虚拟机（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 可以验证以太坊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 是一个用于查找以太坊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 是一个针对以太坊虚拟机（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 语言构建以太坊合约并调试交易。

## 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