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

版本说明

修订内容	时间	修订者	版本号
编写文档	20201229	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档	文档编号	保密级别
Revelation.finance（ADAO）合约审计报告	V1.0	Revelation.finance-ZNNY-20201229	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 12 月 28 日开始到 2020 年 12 月 29 日结束，在此期间针对 **Revelation.finance（ADAO）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，未发现存在明显的安全问题，故综合评定为**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：

条目	描述
Token 名称	Revelation.finance（ADAO）
代码类型	以太坊合约代码
代码语言	solidity
代码地址	https://cn.etherscan.com/address/0x71fbc1d795fcfbca43a3ebf6de0101952f31a410#code

合约文件及哈希：

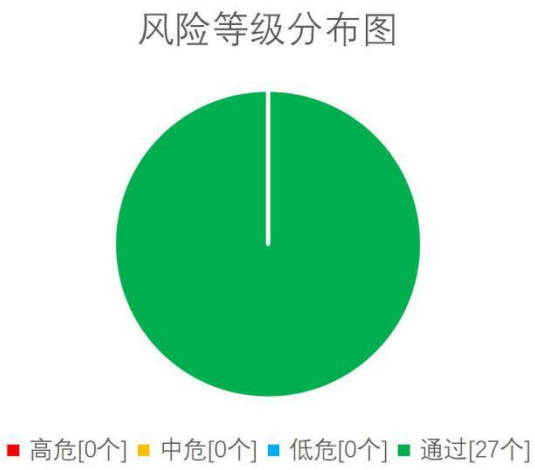
合约文件	MD5
PowerfulERC20.sol	D41D8CD98F00B204E9800998ECF8427E

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:

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	27



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	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。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	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。
	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	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。

	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 代码基本漏洞检测

3.1. 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 0.5.15 以上，不存在该安全问题。

安全建议：无。

3.2. 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。

3.3. 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。

3.4. 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。

3.5. require/assert 的合理使用【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。

3.6. fallback 函数安全【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。

3.7. 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. 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。

3.9. gas 消耗检测【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。

3.10. 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。

3.11. 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。

3.12. 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。

3.13. 访问控制缺陷检测【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。

3.14. 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。

3.15. 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。

3.16. 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。

3.17. 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。

3.18. 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。

3.19. 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，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

3.20. 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。

3.21. 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该安全问题。

安全建议：无。

3.22. 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。

3.23. 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。

3.24. 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。

3.25. 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。

3.26. 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。

3.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方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漏洞。

安全建议:无。

4. 附录 A：合约代码

本次测试代码来源：

```
/**
 *Submitted for verification at Etherscan.io on 2020-11-03
*/

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

// SPDX-License-Identifier: MIT

pragma solidity ^0.7.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable)
        { return msg.sender;
    }
    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

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


pragma solidity ^0.7.0;
/**
 * @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);
}

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


pragma solidity ^0.7.0;

/**
 * @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 message on
```

```
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256)
        { require(b <= a, errorMessage);
        uint256 c = a - b;

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

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

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

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


pragma solidity ^0.7.0;

/**
 * @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-interactio
ns-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal
        { require(address(this).balance >= amount, "Address: insufficient balance");

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

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

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

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

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

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

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

```
            }
        }
}

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


pragma solidity ^0.7.0;



/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20
    { using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

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

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

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

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
```

```
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8)
        { return _decimals;
    }
    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view override returns (uint256)
        { return _totalSupply;
    }
    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override returns (uint256)
        { return _balances[account];
    }
    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
{
        return _allowances[owner][spender];
    }
    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer amount exceeds allowance"));
        return true;
    }
    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
```

```
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

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

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

        _beforeTokenTransfer(sender, recipient, amount);

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

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

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

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
```

```
        require(account != address(0), "ERC20: burn from the zero address");

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

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

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }
    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using
Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Burnable.sol


pragma solidity ^0.7.0;


/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20
    { using SafeMath for uint256;
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }
    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
```

```
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) public virtual {
        uint256 decreasedAllowance = allowance(account, _msgSender()).sub(amount, "ERC20:
burn amount exceeds allowance");

        _approve(account, _msgSender(), decreasedAllowance);
        _burn(account, amount);
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Capped.sol


pragma solidity ^0.7.0;

/**
 * @dev Extension of {ERC20} that adds a cap to the supply of tokens.
 */
abstract contract ERC20Capped is ERC20
    { using SafeMath for uint256;

    uint256 private _cap;

    /**
     * @dev Sets the value of the `cap`. This value is immutable, it can only be
     * set once during construction.
     */
    constructor (uint256 cap_) {
        require(cap_ > 0, "ERC20Capped: cap is 0");
        _cap = cap_;
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view returns (uint256)
        { return _cap;
    }

    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - minted tokens must not cause the total supply to go over the cap.
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual
override {
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) { // When minting tokens require(totalSupply().add(amount)
            <= _cap, "ERC20Capped: cap exceeded");
        }
    }
}

// File: @openzeppelin/contracts/introspection/IERC165.sol


pragma solidity ^0.7.0;

/**
 * @dev Interface of the ERC165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[EIP].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]
     * to learn more about how these ids are created.
```

```
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363.sol


pragma solidity ^0.7.0;


/**
 * @title IERC1363 Interface
 * @dev Interface for a Payable Token contract as defined in
 *  https://eips.ethereum.org/EIPS/eip-1363
 */
interface IERC1363 is IERC20, IERC165 {
    /*
     * Note: the ERC-165 identifier for this interface is 0x4bbee2df.
     * 0x4bbee2df ===
     *     bytes4(keccak256('transferAndCall(address,uint256)')) ^
     *     bytes4(keccak256('transferAndCall(address,uint256,bytes)')) ^
     *     bytes4(keccak256('transferFromAndCall(address,address,uint256)')) ^
     *     bytes4(keccak256('transferFromAndCall(address,address,uint256,bytes)'))
     */

    /*
     * Note: the ERC-165 identifier for this interface is 0xfb9ec8ce.
     * 0xfb9ec8ce ===
     *     bytes4(keccak256('approveAndCall(address,uint256)')) ^
     *     bytes4(keccak256('approveAndCall(address,uint256,bytes)'))
     */

    /**
     *   @notice Transfer tokens from `msg.sender` to another address and then call
`onTransferReceived` on receiver
     * @param to address The address which you want to transfer to
     * @param value uint256 The amount of tokens to betransferred
     * @return true unless throwing
     */
    function transferAndCall(address to, uint256 value) external returns (bool);

    /**
     *   @notice Transfer tokens from `msg.sender` to another address and then call
`onTransferReceived` on receiver
     * @param to address The address which you want to transfer to
     * @param value uint256 The amount of tokens to betransferred
     * @param data bytes Additional data with no specified format, sent in call to `to`
     * @return true unless throwing
     */
    function transferAndCall(address to, uint256 value, bytes calldata data) external returns (bool);

    /**
     * @notice Transfer tokens from one address to another and then call `onTransferReceived` on
receiver
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 The amount of tokens to betransferred
     * @return true unless throwing
     */
    function transferFromAndCall(address from, address to, uint256 value) external returns (bool);

    /**
     * @notice Transfer tokens from one address to another and then call `onTransferReceived` on
receiver
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 The amount of tokens to betransferred
     * @param data bytes Additional data with no specified format, sent in call to `to`
     * @return true unless throwing
     */
    function transferFromAndCall(address from, address to, uint256 value, bytes calldata data)
external returns (bool);

    /**
     * @notice Approve the passed address to spend the specified amount of tokens on behalf of
msg.sender
     * and then call `onApprovalReceived` on spender.
     * Beware that changing an allowance with this method brings the risk that someone may use
both the old
     * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate
this
     *  race condition is to first reduce the spender's allowance to 0 and set the desired value
afterwards:
```

```
        * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        * @param spender address The address which will spend the funds
        * @param value uint256 The amount of tokens to be spent
        */
    function approveAndCall(address spender, uint256 value) external returns (bool);

    /**
        * @notice Approve the passed address to spend the specified amount of tokens on behalf of
msg.sender
        * and then call `onApprovalReceived` on spender.
        * Beware that changing an allowance with this method brings the risk that someone may use
both the old
        * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate
this
        * race condition is to first reduce the spender's allowance to 0 and set the desired value
afterwards:
        * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        * @param spender address The address which will spend the funds
        * @param value uint256 The amount of tokens to be spent
        * @param data bytes Additional data with no specified format, sent in call to `spender`
        */
    function approveAndCall(address spender, uint256 value, bytes calldata data) external returns
(bool);
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363Receiver.sol


pragma solidity ^0.7.0;

/**
  * @title IERC1363Receiver Interface
  * @dev Interface for any contract that wants to support transferAndCall or transferFromAndCall
  *  from ERC1363 token contracts as defined in
  *  https://eips.ethereum.org/EIPS/eip-1363
  */
interface IERC1363Receiver {
    /*
        * Note: the ERC-165 identifier for this interface is 0x88a7ca5c.
        * 0x88a7ca5c === bytes4(keccak256("onTransferReceived(address,address,uint256,bytes)"))
        */

    /**
        * @notice Handle the receipt of ERC1363 tokens
        * @dev Any ERC1363 smart contract calls this function on the recipient
        * after a `transfer` or a `transferFrom`. This function MAY throw to revert and reject the
        * transfer. Return of other than the magic value MUST result in the
        * transaction being reverted.
        * Note: the token contract address is always the message sender.
        * @param operator address The address which called `transferAndCall` or
`transferFromAndCall` function
        * @param from address The address which are token transferred from
        * @param value uint256 The amount of tokens transferred
        * @param data bytes Additional data with no specified format
        * @return `bytes4(keccak256("onTransferReceived(address,address,uint256,bytes)"))`
        *  unless throwing
        */
    function onTransferReceived(address operator, address from, uint256 value, bytes calldata data)
external returns (bytes4);
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363Spender.sol


pragma solidity ^0.7.0;

/**
  * @title IERC1363Spender Interface
  * @dev Interface for any contract that wants to support approveAndCall
  *  from ERC1363 token contracts as defined in
  *  https://eips.ethereum.org/EIPS/eip-1363
  */
interface IERC1363Spender {
    /*
        * Note: the ERC-165 identifier for this interface is 0x7b04a2d0.
        * 0x7b04a2d0 === bytes4(keccak256("onApprovalReceived(address,uint256,bytes)"))
        */

    /**
        * @notice Handle the approval of ERC1363 tokens
        * @dev Any ERC1363 smart contract calls this function on the recipient
        * after an `approve`. This function MAY throw to revert and reject the
        * approval. Return of other than the magic value MUST result in the
        * transaction being reverted.
        * Note: the token contract address is always the message sender.
```

```
     * @param owner address The address which called `approveAndCall` function
     * @param value uint256 The amount of tokens to be spent
     * @param data bytes Additional data with no specified format
     * @return `bytes4(keccak256("onApprovalReceived(address,uint256,bytes)"))`
     *  unless throwing
     */
    function onApprovalReceived(address owner, uint256 value, bytes calldata data) external returns
(bytes4);
}

// File: @openzeppelin/contracts/introspection/ERC165Checker.sol


pragma solidity ^0.7.0;

/**
 * @dev Library used to query support of an interface declared via {IERC165}.
 *
 * Note that these functions return the actual result of the query: they do not
 * `revert` if an interface is not supported. It is up to the caller to decide
 * what to do in these cases.
 */
library ERC165Checker {
    // As per the EIP-165 spec, no interface should ever match 0xffffffff
    bytes4 private constant _INTERFACE_ID_INVALID = 0xffffffff;

    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;

    /**
     * @dev Returns true if `account` supports the {IERC165} interface,
     */
    function supportsERC165(address account) internal view returns (bool) {
        // Any contract that implements ERC165 must explicitly indicate support of
        // InterfaceId_ERC165 and explicitly indicate non-support of InterfaceId_Invalid
        return _supportsERC165Interface(account, _INTERFACE_ID_ERC165) &&
            !_supportsERC165Interface(account, _INTERFACE_ID_INVALID);
    }

    /**
     * @dev Returns true if `account` supports the interface defined by
     * `interfaceId`. Support for {IERC165} itself is queried automatically.
     *
     * See {IERC165-supportsInterface}.
     */
    function supportsInterface(address account, bytes4 interfaceId) internal view returns (bool) {
        // query support of both ERC165 as per the spec and support of _interfaceId
        return supportsERC165(account) &&
            _supportsERC165Interface(account, interfaceId);
    }

    /**
     * @dev Returns true if `account` supports all the interfaces defined in
     * `interfaceIds`. Support for {IERC165} itself is queried automatically.
     *
     * Batch-querying can lead to gas savings by skipping repeated checks for
     * {IERC165} support.
     *
     * See {IERC165-supportsInterface}.
     */
    function supportsAllInterfaces(address account, bytes4[] memory interfaceIds) internal view
returns (bool) {
        // query support of ERC165 itself
        if (!supportsERC165(account)) {
            return false;
        }

        // query support of each interface in _interfaceIds
        for (uint256 i = 0; i < interfaceIds.length; i++) {
            if (!_supportsERC165Interface(account, interfaceIds[i]))
                { return false;
            }
        }

        // all interfaces supported
        return true;
    }

    /**
     * @notice Query if a contract implements an interface, does not check ERC165 support
     * @param account The address of the contract to query for support of an interface
     * @param interfaceId The interface identifier, as specified in ERC-165
     * @return true if the contract at account indicates support of the interface with
     * identifier interfaceId, false otherwise
```

```
     * @dev Assumes that account contains a contract that supports ERC165, otherwise
     * the behavior of this method is undefined. This precondition can be checked
     * with {supportsERC165}.
     * Interface identification is specified in ERC-165.
     */
    function _supportsERC165Interface(address account, bytes4 interfaceId) private view returns
(bool) {
        // success determines whether the staticcall succeeded and result determines
        // whether the contract at account indicates support of _interfaceId
        (bool success, bool result) = _callERC165SupportsInterface(account, interfaceId);

        return (success && result);
    }
    /**
     * @notice Calls the function with selector 0x01ffc9a7 (ERC165) and suppresses throw
     * @param account The address of the contract to query for support of an interface
     * @param interfaceId The interface identifier, as specified in ERC-165
     * @return success true if the STATICCALL succeeded, false otherwise
     * @return result true if the STATICCALL succeeded and the contract at account
     * indicates support of the interface with identifier interfaceId, false otherwise
     */
    function _callERC165SupportsInterface(address account, bytes4 interfaceId)
        private
        view
        returns (bool, bool)
    {
        bytes memory encodedParams = abi.encodeWithSelector(_INTERFACE_ID_ERC165,
interfaceId);
        (bool success, bytes memory result) = account.staticcall{ gas: 30000 }(encodedParams);
        if (result.length < 32) return (false, false);
        return (success, abi.decode(result, (bool)));
    }
}

// File: @openzeppelin/contracts/introspection/ERC165.sol


pragma solidity ^0.7.0;

/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts may inherit from this and call {_registerInterface} to declare
 * their support of an interface.
 */
abstract contract ERC165 is IERC165 {
    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;

    /**
     * @dev Mapping of interface ids to whether or not it's supported.
     */
    mapping(bytes4 => bool) private _supportedInterfaces;

    constructor () {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) public view override returns (bool)
        { return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by
     * `interfaceId`. Support of the actual ERC165 interface is automatic and
     * registering its interface id is not required.
     *
     * See {IERC165-supportsInterface}.
     *
     * Requirements:
     *
     * - `interfaceId` cannot be the ERC165 invalid interface (`0xffffffff`).
     */
    function _registerInterface(bytes4 interfaceId) internal virtual
        { require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
```

```
            _supportedInterfaces[interfaceId] = true;
        }
    }

// File: erc-payable-token/contracts/token/ERC1363/ERC1363.sol


pragma solidity ^0.7.0;




/**
 * @title ERC1363
 * @dev Implementation of an ERC1363 interface
 */
contract ERC1363 is ERC20, IERC1363, ERC165 {
    using Address for address;

    /*
     * Note: the ERC-165 identifier for this interface is 0x4bbee2df.
     * 0x4bbee2df ===
     *     bytes4(keccak256('transferAndCall(address,uint256)')) ^
     *     bytes4(keccak256('transferAndCall(address,uint256,bytes)')) ^
     *     bytes4(keccak256('transferFromAndCall(address,address,uint256)')) ^
     *     bytes4(keccak256('transferFromAndCall(address,address,uint256,bytes)'))
     */
    bytes4 internal constant _INTERFACE_ID_ERC1363_TRANSFER = 0x4bbee2df;

    /*
     * Note: the ERC-165 identifier for this interface is 0xfb9ec8ce.
     * 0xfb9ec8ce ===
     *     bytes4(keccak256('approveAndCall(address,uint256)')) ^
     *     bytes4(keccak256('approveAndCall(address,uint256,bytes)'))
     */
    bytes4 internal constant _INTERFACE_ID_ERC1363_APPROVE = 0xfb9ec8ce;

    // Equals to `bytes4(keccak256("onTransferReceived(address,address,uint256,bytes)"))`
    // which can be also obtained as `IERC1363Receiver(0).onTransferReceived.selector`
    bytes4 private constant _ERC1363_RECEIVED = 0x88a7ca5c;

    // Equals to `bytes4(keccak256("onApprovalReceived(address,uint256,bytes)"))`
    // which can be also obtained as `IERC1363Spender(0).onApprovalReceived.selector`
    bytes4 private constant _ERC1363_APPROVED = 0x7b04a2d0;

    /**
     * @param name Name of the token
     * @param symbol A symbol to be used as ticker
     */
    constructor (string memory name, string memory symbol) ERC20(name, symbol) {
        // register the supported interfaces to conform to ERC1363 via ERC165
        _registerInterface(_INTERFACE_ID_ERC1363_TRANSFER);
        _registerInterface(_INTERFACE_ID_ERC1363_APPROVE);
    }

    /**
     * @dev Transfer tokens to a specified address and then execute a callback on recipient.
     * @param to The address to transfer to.
     * @param value The amount to be transferred.
     * @return A boolean that indicates if the operation was successful.
     */
    function transferAndCall(address to, uint256 value) public override returns (bool)
        { return transferAndCall(to, value, ""); }

    /**
     * @dev Transfer tokens to a specified address and then execute a callback on recipient.
     * @param to The address to transfer to
     * @param value The amount to be transferred
     * @param data Additional data with no specified format
     * @return A boolean that indicates if the operation was successful.
     */
    function transferAndCall(address to, uint256 value, bytes memory data) public override returns
(bool) {
        transfer(to, value);
        require(_checkAndCallTransfer(_msgSender(), to, value, data), "ERC1363:
_checkAndCallTransfer reverts");
        return true;
    }

    /**
     * @dev Transfer tokens from one address to another and then execute a callback on recipient.
```

```
     * @param from The address which you want to send tokens from
     * @param to The address which you want to transfer to
     * @param value The amount of tokens to be transferred
     * @return A boolean that indicates if the operation was successful.
     */
    function transferFromAndCall(address from, address to, uint256 value) public override returns
(bool) {
        return transferFromAndCall(from, to, value, "");
    }
    /**
     * @dev Transfer tokens from one address to another and then execute a callback on recipient.
     * @param from The address which you want to send tokens from
     * @param to The address which you want to transfer to
     * @param value The amount of tokens to be transferred
     * @param data Additional data with no specified format
     * @return A boolean that indicates if the operation was successful.
     */
    function transferFromAndCall(address from, address to, uint256 value, bytes memory data) public
override returns (bool) {
        transferFrom(from, to, value);
        require(_checkAndCallTransfer(from, to, value, data), "ERC1363: _checkAndCallTransfer
reverts");
        return true;
    }
    /**
     * @dev Approve spender to transfer tokens and then execute a callback on recipient.
     * @param spender The address allowed to transfer to
     * @param value The amount allowed to be transferred
     * @return A boolean that indicates if the operation was successful.
     */
    function approveAndCall(address spender, uint256 value) public override returns (bool)
        { return approveAndCall(spender, value, "");
    }
    /**
     * @dev Approve spender to transfer tokens and then execute a callback on recipient.
     * @param spender The address allowed to transfer to.
     * @param value The amount allowed to be transferred.
     * @param data Additional data with no specified format.
     * @return A boolean that indicates if the operation was successful.
     */
    function approveAndCall(address spender, uint256 value, bytes memory data) public override
returns (bool) {
        approve(spender, value);
        require(_checkAndCallApprove(spender, value, data), "ERC1363: _checkAndCallApprove
reverts");
        return true;
    }
    /**
     * @dev Internal function to invoke `onTransferReceived` on a target address
     *  The call is not executed if the target address is not a contract
     * @param from address Representing the previous owner of the given token value
     * @param to address Target address that will receive the tokens
     * @param value uint256 The amount mount of tokens to be transferred
     * @param data bytes Optional data to send along with the call
     * @return whether the call correctly returned the expected magic value
     */
    function _checkAndCallTransfer(address from, address to, uint256 value, bytes memory data)
internal returns (bool) {
        if (!to.isContract())
            { return false;
        }
        bytes4 retval = IERC1363Receiver(to).onTransferReceived(
            _msgSender(), from, value, data
        );
        return (retval == _ERC1363_RECEIVED);
    }
    /**
     * @dev Internal function to invoke `onApprovalReceived` on a target address
     *  The call is not executed if the target address is not a contract
     * @param spender address The address which will spend the funds
     * @param value uint256 The amount of tokens to be spent
     * @param data bytes Optional data to send along with the call
     * @return whether the call correctly returned the expected magic value
     */
    function _checkAndCallApprove(address spender, uint256 value, bytes memory data) internal
returns (bool) {
        if (!spender.isContract())
            { return false;
        }
        bytes4 retval = IERC1363Spender(spender).onApprovalReceived(
            _msgSender(), value, data
        );
```

```
            return (retval == _ERC1363_APPROVED);
        }
}

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


pragma solidity ^0.7.0;

/**
 * @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.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * 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.
 */
abstract 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 () {
                address msgSender = _msgSender();
                _owner = msgSender;
                emit OwnershipTransferred(address(0), msgSender);
        }

        /**
         * @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(_owner == _msgSender(), "Ownable: caller is not the 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 virtual 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 virtual onlyOwner
                { require(newOwner != address(0), "Ownable: new owner is the zero address");
                emit OwnershipTransferred(_owner, newOwner);
                _owner = newOwner;
        }
}

// File: eth-token-recover/contracts/TokenRecover.sol


pragma solidity ^0.7.0;


/**
 * @title TokenRecover
 * @dev Allow to recover any ERC20 sent into the contract for error
 */
contract TokenRecover is Ownable {
```

```
    /**
     * @dev Remember that only owner can call so be careful when use on contracts generated from other contracts.
     * @param tokenAddress The token contract address
     * @param tokenAmount Number of tokens to be sent
     */
    function recoverERC20(address tokenAddress, uint256 tokenAmount) public onlyOwner
        { IERC20(tokenAddress).transfer(owner(), tokenAmount);
    }
}
// File: contracts/service/ServiceReceiver.sol


pragma solidity ^0.7.0;

/**
 * @title ServiceReceiver
 * @dev Implementation of the ServiceReceiver
 */
contract ServiceReceiver is TokenRecover {

    mapping (bytes32 => uint256) private _prices;

    event Created(string serviceName, address indexed serviceAddress);

    function pay(string memory serviceName) public payable {
        require(msg.value == _prices[_toBytes32(serviceName)], "ServiceReceiver: incorrect price");

        emit Created(serviceName, _msgSender());
    }

    function getPrice(string memory serviceName) public view returns (uint256)
        { return _prices[_toBytes32(serviceName)];
    }
    function setPrice(string memory serviceName, uint256 amount) public onlyOwner {
        _prices[_toBytes32(serviceName)] = amount;
    }

    function withdraw(uint256 amount) public onlyOwner
        { payable(owner()).transfer(amount);
    }

    function _toBytes32(string memory serviceName) private pure returns (bytes32)
        { return keccak256(abi.encode(serviceName));
    }
}
// File: contracts/service/ServicePayer.sol


pragma solidity ^0.7.0;

/**
 * @title ServicePayer
 * @dev Implementation of the ServicePayer
 */
contract ServicePayer {

    constructor (address payable receiver, string memory serviceName) payable
        { ServiceReceiver(receiver).pay{value: msg.value}(serviceName);
    }
}
// File: contracts/token/ERC20/PowerfulERC20.sol


pragma solidity ^0.7.0;




/**
 * @title PowerfulERC20
 * @dev Implementation of the PowerfulERC20
 */
contract PowerfulERC20 is ERC20Capped, ERC20Burnable, ERC1363, TokenRecover, ServicePayer {
```

```
    // indicates if minting is finished
    bool private _mintingFinished = false;

    /**
     * @dev Emitted during finish minting
     */
    event MintFinished();

    /**
     * @dev Tokens can be minted only before minting finished.
     */
    modifier canMint() {
        require(!_mintingFinished, "PowerfulERC20: minting is finished");
        _;
    }

    constructor (
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 cap,
        uint256 initialBalance,
        address payable feeReceiver
    ) ERC1363(name, symbol) ERC20Capped(cap) ServicePayer(feeReceiver, "PowerfulERC20")
payable {
        _setupDecimals(decimals);
        _mint(_msgSender(), initialBalance);
    }

    /**
     * @return if minting is finished or not.
     */
    function mintingFinished() public view returns (bool)
        { return _mintingFinished;
    }

    /**
     * @dev Function to mint tokens.
     * @param to The address that will receive the minted tokens
     * @param value The amount of tokens to mint
     */
    function mint(address to, uint256 value) public canMint onlyOwner {
        _mint(to, value);
    }

    /**
     * @dev Function to stop minting new tokens.
     */
    function finishMinting() public canMint onlyOwner {
        _mintingFinished = true;

        emit MintFinished();
    }

    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual
override(ERC20, ERC20Capped) {
        super._beforeTokenTransfer(from, to, amount);
    }
}
```

5. 附录 B：安全风险评级标准

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：智能合约安全审计工具简介

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 知道创宇区块链安全审计人员专用工具包

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。

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587
邮　　箱 sec@knownsec.com
官　　网 www.knownsec.com
地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509