KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: **KnownSec blockchain security research team**

Release notes

Revised content	Time	Revised by	Version number
Written document	20201029	KnownSec blockchain security research team	V1.0

Document information

Document name	Document version number	Document number	Confidentiality level
Revelation.finance(ADAO) **Smart Contract Audit Report**	V1.0	Revelation.finance-ZNNY-20201229	**Open project Team**

The statement

KnownSec only issues this report on the facts that have occurred or exist before the issuance of this report, and shall assume the corresponding responsibility therefor.KnownSec is not in a position to judge the security status of its smart contract and does not assume responsibility for the facts that occur or exist after issuance.The security audit analysis and other contents of this report are based solely on the documents and information provided by the information provider to KnownSec as of the issuance of this report.KnownSec assumes that the information provided was not missing, altered, truncated or suppressed.If the information provided is missing, altered, deleted, concealed or reflected in a way inconsistent with the actual situation, KnownSec shall not be liable for any loss or adverse effect caused thereby.

Directory

1. Review

The effective test time of this report is from December 28, 2020 to December 29, 2020. During this period, the security and standardization of the Revelation.finance （ADAO） smart contract code will be audited and used as the statistical basis for the report.

In this test, it was knownsec engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (see Chapter 3), and no obvious security problems were found, so the comprehensive assessment was passed.

The results of this smart contract security audit: Pass.

Since this test is conducted in a non-production environment, all codes are updated, the test process is communicated with the relevant interface personnel, and relevant test operations are carried out under the control of operational risks, so as to avoid production and operation risks and code security risks in the test process.

The target information of this test:

entry	description
Token name	Revelation.finance（ADAO）
Code type	Ethereum contract code
Code language	solidity
Code address	https://cn.etherscan.com/address/0x71fbc1d795fcfbca43a3ebf6de0101952f31a410#code

Contract Documents and Hash:

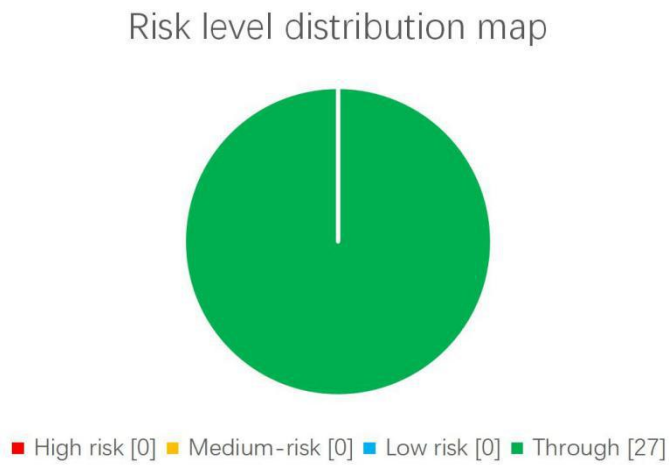
The contract documents	MD5
PowerfulERC20.sol	D41D8CD98F00B204E9800998ECF8427E

2. Code vulnerability analysis

2.1 Vulnerability level distribution

This vulnerability risk is calculated by level:

Statistics on the number of security risk levels			
High Risk	Medium Risk	Low Risk	Pass
0	0	0	27



2.2 Summary of audit results

Audit results			
Audit item	Audit content	status	description
Basic code vulnerability detection	Compiler version security	pass	After testing, there are no safety issues.
	Redundant code	pass	After testing, there are no safety issues.
	Use of safe arithmetic library	pass	After testing, there are no safety issues.
	Not recommended encoding	pass	After testing, there are no safety issues.
	Reasonable use of require/assert	pass	After testing, there are no safety issues.
	fallback function safety	pass	After testing, there are no safety issues.
	tx.orgin authentication	pass	After testing, there are no safety issues.
	Owner permission control	pass	After testing, there are no safety issues.
	Gas consumption detection	pass	After testing, there are no safety issues.
	call injection attack	pass	After testing, there are no safety issues.
	Low-level function safety	pass	After testing, there are no safety issues.
	Vulnerability of additional token issuance	pass	After testing, there are no safety issues.
	Access control defect detection	pass	After testing, there are no safety issues.
	Numerical overflow detection	pass	After testing, there are no safety issues.
	Arithmetic accuracy error	pass	After testing, there are no safety issues.
	Wrong use of random number detection	pass	After testing, there are no safety issues.
	Unsafe interface use	pass	After testing, there are no safety issues.

	Variable coverage	pass	After testing, there are no safety issues.
	Uninitialized storage pointer	pass	After testing, there are no safety issues.
	Return value call verification	pass	After testing, there are no safety issues.
	Transaction order dependency detection	pass	After testing, there are no safety issues.
	Timestamp dependent attack	pass	After testing, there are no safety issues.
	Denial of service attack detection	pass	After testing, there are no safety issues.
	Fake recharge vulnerability detection	pass	After testing, there are no safety issues.
	Reentry attack detection	pass	After testing, there are no safety issues.
	Replay attack detection	pass	After testing, there are no safety issues.
	Rearrangement attack detection	pass	After testing, there are no safety issues.

3. Basic code vulnerability detection

3.1. Compiler version security [Pass]

Check whether a safe compiler version is used in the contract code implementation.

Test result: After testing, the compiler version 0.5.15 is formulated in the smart contract code, and there is no such security issue.

Safety advice: None.

3.2. Redundant code [Pass]

Check whether the contract code implementation contains redundant code.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.3. Use of safe arithmetic library [Pass]

Check whether the SafeMath safe arithmetic library is used in the contract code implementation

Test result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Safety advice: None.

3.4. Not recommended encoding [Pass]

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.5. Reasonable use of require/assert [Pass]

Check the rationality of the use of require and assert statements in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.6. fallback function safety [Pass]

Check whether the fallback function is used correctly in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.7. tx.orgin authentication [Pass]

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks like phishing.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.8. Owner permission control [Pass]

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.9. Gas consumption detection [Pass]

Check whether the consumption of gas exceeds the maximum block limit

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.10. call injection attack [Pass]

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Detection result: After detection, the smart contract does not use the call function, and this vulnerability does not exist.

Safety advice: None.

3.11. Low-level function safety [Pass]

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.12. Vulnerability of additional token issuance [Pass]

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.13. Access control defect detection [Pass]

Different functions in the contract should set reasonable permissions

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.14. Numerical overflow detection[Pass]

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Test result: After testing, the security problem does not exist in the smart

contract code.

Safety advice: None.

3.15. Arithmetic accuracy error [Pass]

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.16. Wrong use of random number detection [Pass]

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners. These random numbers are predictable to a certain

extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.17. Unsafe interface use [Pass]

Check whether unsafe interfaces are used in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.18. Variable coverage [Pass]

Check whether there are security issues caused by variable coverage in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.19. Uninitialized storage pointer [Pass]

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Test result: After testing, the smart contract code does not use structure, and there is no such problem.

Safety advice: None.

3.20. Return value call verification [Pass]

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

There are transfer(), send(), call.value() and other currency transfer methods in Solidity, which can all be used to send Ether to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be passed for calling (can be By passing in the gas_value parameter to limit), it cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which

may lead to unexpected results due to Ether sending failure.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.21. Transaction order dependency detection [Pass]

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

Test result: After testing, the security issue does not exist in the smart contract code.

Safety advice: None.

3.22. Timestamp dependent attack [Pass]

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it is only necessary to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to

him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.23. Denial of service attack detection [Pass]

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.24. Fake recharge vulnerability detection [Pass]

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, it enters the else logic part and returns false, and finally no exception is thrown. We

believe that only if/else this kind of gentle judgment method is an imprecise coding method in the scene of sensitive functions such as transfer.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.25. Reentry attack detection [Pass]

Re-entry vulnerability is the most famous Ethereum smart contract vulnerability, which once led to the fork of Ethereum (The DAO hack).

The call.value() function in Solidity consumes all the gas it receives when it is used to send Ether. When the call.value() function is called to send Ether before it actually reduces the balance of the sender's account, There is a risk of reentry attacks.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

3.26. Replay attack detection [Pass]

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts. .

Detection result: After detection, the smart contract does not use the call function, and this vulnerability does not exist.

Safety advice: None.

3.27. Rearrangement attack detection [Pass]

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Test result: After testing, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

4. Appendix A: Contract code

Source code for this test:

```
/**
 *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-enc
oding-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. Appendix B: Vulnerability risk rating criteria

Smart contract vulnerability rating standards	
Vulnerability rating	Vulnerability rating description
High-risk vulnerabilities	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.;
Mid-risk vulnerability	Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.;
Low-risk vulnerabilities	Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.

6. Appendix C: Introduction to vulnerability testing tools

6.1. Manticore

A Manticore is a symbolic execution tool for analyzing binary files and smart contracts. A Manticore consists of a symbolic Ethereum virtual machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of the Solarium body.It also incorporates Ethersplay, a Bit of Traits of Bits visual disassembler for EVM bytecode, for visual analysis. Like binaries, Manticore provides a simple command-line interface and a Python API for analyzing EVM bytecode.

6.2. Oyente

Oyente is a smart contract analysis tool that can be used to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and so on.More conveniently, Oyente's design is modular, so this allows power users to implement and insert their own inspection logic to check the custom properties in their contracts.

6.3. securify. Sh

Securify verifies the security issues common to Ethereum's smart contracts, such as unpredictability of trades and lack of input verification, while fully automated and analyzing all possible execution paths, and Securify has a specific language for identifying vulnerabilities that enables the securities to focus on current security and other reliability issues at all times.

6.4. Echidna

Echidna is a Haskell library designed for fuzzy testing EVM code.

6.5. MAIAN

MAIAN is an automated tool used to find holes in Ethereum's smart contracts. MAIAN processes the bytecode of the contract and tries to set up a series of transactions to find and confirm errors.

6.6. ethersplay

Ethersplay is an EVM disassembler that includes correlation analysis tools.

6.7. IDA - evm entry

Ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

6.8. want - ide

Remix is a browser-based compiler and IDE that allows users to build ethereum contracts and debug transactions using Solarium language.

6.9. KnownSec Penetration Tester kit

KnownSec penetration tester's toolkit, developed, collected and used by KnownSec penetration tester engineers, contains batch automated testing tools, self-developed tools, scripts or utilization tools, etc. dedicated to testers.

Beijing Knownsec Information Tech.CO.,LTD

Telephone	+86(10)400 060 9587
Email	sec@knownsec.com
Official website	www.knownsec.com
Address	2509, block T2-B, WangJing SOHO, Chaoyang District, Beijing